Modélisation et simulation des systèmes de production : une approche orientée-objets

Xiaojun Ye

No d'ordre 94 Année 1994

THESE

présentée devant

L'INSTITUT NATIONAL DES SCIENCES APPLIQUEES DE LYON

pour obtenir

LE GRADE DE DOCTEUR

SPECIALITE: INGENIERIE INFORMATIQUE

par
Xiaojun YE

(Ingénieur en Mécanique Industrielle)

Modélisation et Simulation des Systèmes de Production: une Approche Orientée-Objets

---

Soutenue le 29 juin 1994 devant la Commission d'Examen

Jury MM.	Gérard BEL	
	Joël FAVREL	
	Gia Toan NGUYEN	Rapporteur
	Georges JAVEL	Rapporteur
	Jean-Paul KIEFFER	Rapporteur
	Albert MATHON	

No d'ordre 94 Année 1994

THESE

présentée devant

L'INSTITUT NATIONAL DES SCIENCES APPLIQUEES DE LYON

pour obtenir

LE GRADE DE DOCTEUR

SPECIALITE: INGENIERIE INFORMATIQUE

par
Xiaojun YE

(Ingénieur en Mécanique Industrielle)

Modélisation et Simulation des Systèmes de Production: une Approche Orientée-Objets

---

Soutenue le 29 juin 1994 devant la Commission d'Examen

Jury MM. Gérard BEL
Joël FAVREL
Gia Toan NGUYEN — Rapporteur
Georges JAVEL — Rapporteur
Jean-Paul KIEFFER — Rapporteur
Albert MATHON

NOVEMBRE 1993

INSTITUT NATIONAL DES SCIENCES APPLIQUEES DE LYON

Directeur : J.ROCHAT

Professeurs :

S.AUDISIO	PHYSICOCHIMIE INDUSTRIELLE
J.C.BABOUX	TRAIT. SIGNAL ULTRASONS
J.BAHUAUD	MECANIQUE DES SOLIDES
B.BALLAND	PHYSIQUE DE LA MATIERE
G.BAYADA	CENTRE DE MATHEMATIQUES
C.BERGER (Melle)	PHYSIQUE INDUSTRIELLE
M.BETEMPS	AUTOMATIQUE INDUSTRIELLE
C.BOISSON	VIBRATIONS ACOUSTIQUES
M.BOIVIN	MECANIQUE DES SOLIDES
H.BOTTA	GENIE CIVIL ET URBANISME (METHODES)
G.BOULAYE	INGENIERIE DES SYSTEMES D'INFORMATION
J.BRAU	EQUIPEMENT DE L'HABITAT
M.BRUNET	MECANIQUE DES SOLIDES
J.C.BUREAU	THERMOCHIMIE MINERALE
J.P.CHANTE	COMPOSANTS DE PUISSANCE ET APPLICATIONS
M.CHEVRETON	ETUDE DES MATERIAUX
B. CHOCAT	METHODES
B.CLAUDEL	CHIMIE PHYSIQUE APPLIQUEE ET ENVIRONNEMENT
L.CRONENBERGER	CHIMIE BIOLOGIQUE
M.DIOT	THERMOCHIMIE MINERALE
A.DOUTHEAU	CHIMIE ORGANIQUE
B.DUPERRAY	CHIMIE BIOLOGIQUE
H.EMPTOZ	MOD.SYST.ET REC.DES FORMES
C.ESNOUF	GEMPPM*
L.EYRAUD	GENIE ELECTRIQUE ET FERROELECTRICITE
G.FANTOZZI	GEMPPM*
J.FAUCHON	CONCEPTION ET ANALYSE SYSTEMES MECA.
J.FAVREL	INFORMATIQUE DES SYST. DE PROD. INDUS.
Y.FETIVEAU	GENIE ELECTRIQUE ET FERROELECTRICITE
L.FLAMAND	MECANIQUE DES CONTACTS
P.FLEISCHMANN	GEMPPM*
A.FLORY	INGENIERIE DES SYSTEMES D'INFORMATION
R.FOUGERES	GEMPPM*
L.FRECON	DEVELOP. LANGAGES INFORMAT. AVANCES
R.GAUTHIER	PHYSIQUE DE LA MATIERE
M.GERY	GCU (EQUIPEMENT DE L'HABITAT)
G.GIMENEZ	TRAITEMENT DU SIGNAL ET ULTRASONS
P.GOBIN	GEMPPM*
P.GONNARD	GENIE ELECTRIQUE ET FERROELECTRICITE
R.GOUTTE	TRAITEMENT DU SIGNAL ET ULTRASONS
G.GRANGE	GENIE ELECTRIQUE
G.GUENIN	GEMPPM*
G.GUILLOT	PHYSIQUE DE LA MATIERE
A.GUINET	INFORMATIQUE DES SYST.DE PROD.INDUS.
C.GUITTARD	DEVELOP.LANGAGES INFORMAT.AVANCES
J.L.GUYADER	VIBRATIONS-ACOUSTIQUE
J.JOUBERT	GENIE MECANIQUE
J.F.JULLIEN	BETONS ET STRUCTURES
A.JUTARD	AUTOMATIQUE INDUSTRIELLE
R.KASTNER	GEOTECHNIQUE
H.KLEIMANN	GENIE ELECTRIQUE ET FERROELECTRICITE
J.KOULOUMDJIAN	INGENIERIE DES SYSTEMES D'INFORMATION
M.LAGARDE	CHIMIE BIOLOGIQUE
M.LALANNE	MECANIQUE DES STRUCTURES
A.LALLEMAND	ENERGETIQUE ET AUTOMATIQUE
M.LALLEMAND (Mme)	ENERGETIQUE ET AUTOMATIQUE

./.

(NOVEMBRE 1993)

P.LAREAL	GENIE CIVIL ET URBANISME GEOTECHNIQUE)
A.LAUGIER	PHYSIQUE DE LA MATIERE
CH.LAUGIER	PHYSIOLOGIE ET PHARMACODYNAMIE
P.LEJEUNE	GENETIQUE MOLECULAIRE DES MICROORGANISMES
C.LESUEUR	VIBRATIONS-ACOUSTIQUE
Y.MARTINEZ	INFORMATIQUE DES SYST. DE PROD. INDUST.
C.MARTY	CONCEPTION ET ANALYSE SYSTEMES MECA.
J. MERLIN	GEMPPM*
H.MAZILLE	PHYSICOCHIMIE INDUSTRIELLE
M.MIRAMOND	METHODES
N.MONGEREAU	GENIE CIVIL (GEOTECHNIQUE)
R.MOREL	MECANIQUE DES FLUIDES ET THERMIQUES
P.NARDON	BIOLOGIE
A.NAVARRO	CHIMIE PHYSIQUE APPLIQUEE ET ENVIRON.
M.OTTERBEIN	CHIMIE PHYSIQUE APPLIQUEE ET ENVIRON.
J.P.PASCAULT	MATERIAUX MACROMOLECULAIRES
J.PERA	SOLIDES ET MATERIAUX MINERAUX
G.PERACHON	THERMOCHIMIE MINERALE
M.PERDRIX	TRAITEMENT DU SIGNAL ET ULTRASONS
J.PEREZ	GEMPPM*
P.PINARD	PHYSIQUE DE LA MATIERE ET PHYSIQUE INDUSTRIELLE
D.PLAY	CONCEPTION ET ANALYSE SYSTEMES MECA.
P.PREVOT	INFORMATIQUE DES SYST. DE PROD. INDUST.
R.REYNAUD	ENERGETIQUE ET AUTOMATIQUE
J.M.REYNOUARD	BETONS ET STRUCTURES
M.RICHARD	ENERGETIQUE ET AUTOMATIQUE
E.RIEUTORD	MECANIQUE DES FLUIDES ET THERMIQUE
J.ROBERT-BAUDOUY (Mme)	GENETIQUE MOLECULAIRE DES MICROORGANISMES
J.ROBIN	PHYSICOCHIMIE INDUSTRIELLE
D.ROUBY	GEMPPM*
J.F.SACADURA	MECANIQUE DES FLUIDES ET THERMIQUE
H.SAUTEREAU	MATERIAUX MACROMOLECULAIRES
S.SCAVARDA	AUTOMATIQUE INDUSTRIELLE
F.STOEBER	GENETIQUE MOLECULAIRE DES MICROORGANISMES
M.TROCCAZ	GENIE ELECTRIQUE ET FERROELECTRICITE
J.TUSET	SOLIDES ET MATERIAUX MINERAUX
R.UNTERREINER	TRAITEMENT DU SIGNAL ET ULTRASONS
P.VERMANDE	CHIMIE PHYSIQUE APPLIQUEE ET ENVIRON.
J.VERON	CHIMIE PHYSIQUE APPLIQUEE ET ENVIRON.
A.VINCENT	GEMPPM*
P.VUILLERMOZ	PHYSIQUE DE LA MATIERE

Directeurs de recherche C.N.R.S. :

P.CLAUDY	THERMOCHIMIE MINERALE
M.MURAT	GEMPPM*
A.NOUAILHAT	PHYSIQUE DE LA MATIERE
M.A.MANDRAND (Mme)	GENETIQUE MOLECULAIRE DES MICROORGANISMES

Directeurs de recherche I.N.R.A. :

G.BONNOT	BIOLOGIE
S.GRENIER	BIOLOGIE
Y.MENEZO	BIOLOGIE

Directeurs de recherche I.N.S.E.R.M. :

A-F. PRIGENT (Mme)	CHIMIE BIOLOGIQUE
N.SARDA (Mme)	CHIMIE BIOLOGIQUE

* GROUPE D'ETUDE METALLURGIE PHYSIQUE ET PHYSIQUE DES MATERIAUX

à mes parents

REMERCIEMENTS

Le travail présenté dans ce mémoire a été réalisé au Département Stratégie du Développement de l'Ecole Nationale Supérieure des Mines de Saint-Etienne.

Je tiens, tout d'abord, à remercier Monsieur Albert MATHON, professeur et Directeur des Etudes de l'Ecole Nationale Supérieure des Mines de Saint-Etienne, ancien Directeur du Département Stratégie du Développement, qui m'a accueilli dans son équipe et qui a dirigé cette étude. Qu'il soit également remercié pour la confiance dont il a bien voulu faire preuve à mon égard, sa disponibilité et son aide tant scientifique qu'administrative.

J'exprime toute ma gratitude à Monsieur Joël FAVREL, professeur à l'Institut National des Sciences Appliquées de Lyon, Directeur de l'Atelier Interuniversitaire Productique de Lyon, de m'avoir accueilli dans son Groupe de Recherche en Analyse de Système et Productique pour mon DEA, et de sa disponibilité, sa caution scientifique, ainsi que son intérêt porté aux travaux de ce mémoire qui ont constitué pour moi un soutien important.

Que Monsieur Gia Toan NGUYEN, directeur de recherche à l'INRIA Rhône-Alpes, soit vivement remercié pour avoir accepté de rapporter sur ce travail et pour sa participation au jury. Je profite de cette occasion pour le remercier de l'attention que le Groupe de Travail "Objets" du Pôle Productique, qu'il anime, a porté à mon travail.

Je tiens à exprimer ma gratitude à Monsieur Jean-Paul KIEFFER, professeur à l'Université d'Aix-Marseille, pour l'honneur qu'il me fait en acceptant de rapporter sur ce travail en dépit des charges multiples qu'il assume.

A Monsieur Georges JAVEL, professeur à l'IUT de Nantes, pour l'intérêt qu'il a manifesté pour cette recherche en acceptant d'en être rapporteur.

Je remercie également Monsieur Gérard BEL, Maître de Recherche à l'ONERA-CERT de Toulouse, pour l'honneur qu'il me fait en acceptant de participer à mon jury de thèse.

Je remercie les membres de l'équipe Etude et Modélisation des Systèmes Industriels pour leur aide diverse, en particulier Messieurs Bertrand JULLIEN, Lucien VINCENT et Saïd MIRY,

pour les discussions fructueuses qui m'ont permis de progresser dans ce travail. Je tiens à remercier également Messieurs Redouane SENOUNE et François LAURENT et les membres de l'équipe qui ont lu et corrigé, sur le fond et la forme, tout ou partie de cette thèse.

Mes remerciements s'adressent enfin à tous ceux qui au sein de l'ex-Département Stratégie du Développement ont su créer et entretenir une atmosphère de sympathie et de confiance dont j'ai grandement bénéficié. Je suis très touché par la gentillesse de tous ceux, en particulier M^elle^ Bernadette ZOLD, M^elle^ Zahia MAZER et Monsieur André LOUBET, qui m'ont donné un coup de main, amicalement et généreusement, pendant la préparation ou le jour de soutenance de cette thèse.

Et ce n'est pas cette occasion qui rendra faciles à décrire mes sentiments envers ma famille, ma copine et mes copains à l'Ecole et ailleurs!

RESUME

L'approche objet permet des applications plus évoluées et plus fiables et des développements spécifiques moins coûteux et évolutifs. Les objectifs de ce travail sont, d'une part, de contribuer à la conceptualisation complète de modèles de simulation à objet et d'autre part, de les implémenter en utilisant des techniques de programmation concurrente. Après une présentation, au chapitre I, des concepts des systèmes de production et de leur gestion, nous avons évalué, au chapitre II, les différents modèles de structure et de simulation pour les systèmes de production. Le chapitre III propose une démarche d'analyse pour identifier des classes d'objets en cinq types du domaine: physiques, rôles, incidents, interactions et spécifications. Chacune de ces classes est spécifiée par quatre modèles: communication, information, transition d'état et processus. Dans le chapitre IV, nous avons conceptualisé une architecture générale des objets actifs, une plateforme de simulation à objets concurrents et des classes d'objets sémantiques tels que les transactions, les moyens de production et les décisions pour l'établissement des modèles de simulation de production. Nous avons illustré, au chapitre V, l'implémentation des coopérations spatiales et temporelles entre objets concurrents dans la simulation avec des concepts processus "légers" basés sur l'outil Meijin++.

MOTS-CLES

Système Production, Modélisation, Simulation, Orienté Objet, Programmation Parallèle, Processus Communicants

ABSTRACT

The object-oriented approach allows the development of complex and reliable applications with less effort than with classical approaches. The objectives of this research are, on the one hand, to propose a complete conceptualization of object-oriented simulation models and, on the other hand, to implement them by using concurrent programming techniques. After the presentation of the manufacturing systems and their management in chapter I, we classify the different structure and simulation models for production systems in chapter II. In chapter III, we propose an analysis method to identify object classes by five domain types: physical, role, incident, interaction and specification. Each of these classes is specified by four models: communication, information, state transition and process. A general architecture of active objects and of simulation platform and the principal semantic object classes (like transactions, production facilities and decision objects) to establish production simulation models are presented in chapter IV. In chapter V we illustrate the implementation of spatial and timing coordination between concurrent objects in the simulation by using the concept of light-weight processes based on the Meijin++ tool.

KEYWORDS

Production System, Modeling, Simulation, Object-Oriented, Parallel Programming, Communicating Process

Table Matière

Chapitre II Modélisation et Simulation des Systèmes de Production

Chapitre III Analyse des Systèmes de Production par l'Approche Objet

Chapitre IV Conception d'un Modèle de Simulation des Systèmes de Production par l'Approche Objet

Chapitre V Simulation Concurrente par l'Approche Objet

INTRODUCTION

La conception, l'apprentissage ou le développement des systèmes de production industriels impliquent des investissements humains et matériels souvent très coûteux. L'intégration de la simulation dans le domaine industriel permet d'abaisser considérablement ces coûts.

Dans la phase de conception d'un système de production, la simulation permet de tester puis de valider l'architecture de l'atelier et d'expérimenter à moindre frais les différents systèmes de conduite envisageables pour une production donnée. Lors de l'apprentissage d'un pilote de conduite d'atelier, la simulation permet à celui-ci d'acquérir une certaine expérience sans risque d'accident ni de dégât matériel, toujours onéreux et parfois catastrophiques. Enfin, dans les phases de développement ou de réorganisation du système de production, les essais de validation et de conception des commandes pourront être entrepris sans nuire à l'installation actuelle ni à son fonctionnement.

Or, le développement d'un modèle de simulation est une activité coûteuse en temps. Dans l'approche traditionnelle, le développeur du modèle doit faire appel à son art et à son expérience pour produire une description algorithmique des activités et des événements d'un projet. Cela implique une grande partie d'efforts répétitifs consacrés au développement de logiciels spécifiques. Une fois implémenté et utilisé, ce modèle est très rarement réutilisable ou réexploitable. De nouvelles approches et de nouveaux types de modèles sont donc nécessaires pour résoudre efficacement ce genre de problèmes (réutilisabilité et flexibilité du modèle).

L'enjeu informatique actuel des systèmes de production est de trouver des approches regroupant concepts, méthodes et outils, dans le domaine de l'intelligence artificielle et du génie logiciel, qui leur permettent d'adapter leurs activités et ainsi leurs applications aux perpétuelles fluctuations de leur environnement économique.

En effet, si nous considérons le système de production, non pas comme une organisation monolithique, mais comme un ensemble d'unités concurrentes de traitement et de communication d'information, nous obtenons une organisation distribuée, plus dynamique, d'un caractère nouveau. Cette organisation est nécessaire à l'adaptation des systèmes de production à la constante mutation de son environnement économique. Ainsi les solutions des problèmes dépendent de la maîtrise des flux d'information manipulés par ces unités de traitement et de communication d'information.

Une démarche cohérente d'informatisation de la fonction production de l'entreprise manufacturière se doit d'être une démarche globale. Aussi, le système d'information du système de production ne peut être "pensé" que globalement. L'informatisation d'une activité ne doit donc pas être réalisée indépendamment des autres activités, et en généralisant, indépendamment de la logique qui veut que le système ne produise que pour répondre aux besoins des clients.

Une telle démarche se heurte à des problèmes complexes et évolutifs, qui ne peuvent être appréhendés autrement que par une démarche de modélisation et de simulation. Les différents outils de modélisation pour la simulation jusqu'à présent utilisés, ont cependant été développés pour des applications spécifiques, illustrant surtout le cloisonnement qui existe entre les activités du système de production.

Cette voie n'est aujourd'hui plus acceptable et met en avant le besoin d'un outil de modélisation plus général. C'est le concept objet, entité fondamentale des systèmes à objets, qui constitue la première partie du centre d'intérêt de ce mémoire et qui peut raisonnablement prétendre apporter cette généralité. Le concept objet nous permet de faire du système de production un modèle plus fidèle à la réalité et à la pensée humaine. Un modèle de simulation est surtout caractérisé par son aspect dynamique; un objet doit non seulement avoir une frontière d'encapsulation, mais également avoir une frontière d'interaction avec les autres processus. Le concept de programmation concurrente pour la simulation constitue la seconde partie de ce mémoire, il permet d'exprimer de manière plus commode et plus flexible la dynamique ou le comportement temporel des systèmes de production.

Avec le besoin d'intégrer, c'est au travers des étapes traditionnelles de conception, de fabrication et de gestion que l'entreprise fixe un pôle d'intégration. Nous nous sommes intéressés à la gestion de production pour la raison principale que ce pôle d'intégration présente aujourd'hui les résultats les plus décevants en matière de traitement et de communication automatisés d'informatisation.

L'informatisation de la gestion de production témoigne des enjeux de l'intégration. Mise en oeuvre à travers des logiciels de gestion de production assistée par ordinateur (GPAO), la gestion de production ne contribue pas ou encore très mal à la compétitivité de l'entreprise, ramenant ainsi l'outil informatique à un simple outil d'exécution. L'intégration de la simulation à la gestion nous permet d'évaluer plus rapidement et plus précisément les performances de différents plans de production ou stratégies d'ordonnancement dans un système de production afin de mieux l'exploiter. Alors que les méthodes de gestion de production se compliquent, rendant incontournable l'utilisation d'un outil informatique plus performant et plus puissant, il y

a lieu de s'interroger sur l'opportunité de traiter les problèmes liés à la simulation des systèms de gestion de production avec des approches nouvelles, ce que nous avons fait avec l'approche objet et l'approche de la programmation concurrente (ou programmation parallèle).

Au même titre que l'on parle de flexibilité de production, on peut parler de flexibilité de gestion, et plus généralement de flexibilité des applications qui contribuent à la compétitivité de l'entreprise. Le modèle objets est à notre sens le moyen d'obtenir la flexibilité et la fiabilité de modèle de production, et la programmation concurrente avec une interface conviviale est le moyen d'obtenir la flexibilité et la dynamique de la gestion de production.

C'est en ce sens que nous avons effectué les travaux présentés dans ce mémoire et par là même que nous faisons des propositions. Nous avons en particulier organisé ce mémoire en cinq chapitres pour nous amener, problèmes après problèmes, à proposer une approche objet dédiée à l'analyse des systèmes de production, une approche processus dédiée à la conception d'un modèle de simulation des flux, une approche programmation concurrente dédiée à l'implémentation de modèle de simulation, ainsi qu'une plate-forme qui va intégrer un module de la simulation dans la gestion de production comme un outil d'aide à la décision.

Le premier chapitre est consacré à une présentation générale des systèmes de production et de leur gestion. Nous y présentons les méthodes de gestion qui, à notre avis, sont les plus marquantes en planification et ordonnancement de la production. Cependant, ces méthodes ou logiques de gestion de production, informatisées dans les logiciels de GPAO, s'avèrent d'une utilisation rigide du fait de l'impossibilité de les adapter à la spécificité, ce qui nous amènent à reconsidérer les modèles de représentation des systèmes de production.

Le chapitre II présente les méthodes représentatives d'analyse et de conception des systèmes de production: les modèles de structure correspondant aux méthodes SADT, MERISE, GRAI et CIMOSA et les modèles de simulation. Nous décrivons les principaux concepts sur lesquels ces méthodes s'appuient, ainsi que les différentes étapes d'analyse et de conception des systèmes de production. Enfin, nous effectuons des analyses critiques de ces méthodes par rapport à la flexibilité du problème et à la flexibilité de sa représentation pour introduire les trois chapitres suivants: utilisation de l'approche objets pour analyser, concevoir et implémenter les modèles de simulation des systèmes de production.

Les objets des systèmes de production sont identifiés et spécifiés dans le chapitre III. Deux étapes d'analyse sont proposées: une étape d'analyse du domaine qui établit le contexte principal dans lequel le système sera construit et une étape d'analyse de l'application qui se concentre sur le domaine d'intérêt afin de construire un cahier des charges pour une application

particulière. Les types d'objets et leurs modèles correspondants: modèles de communication, modèles de transition d'états et modèles d'information sont présentés

Dans le chapitre IV, nous appuyant sur les concepts d'objets autonomes et actif et d'objet passif, nous avons conceptualisé une architecture générale d'un objet actif et conçu une architecture d'un modèle ouvert et flexible et l'intégration des décisions dans le modèle de simulation. Les classes d'objets sémantiques pour les systèmes de production, les classes d'objets d'application et de résolution sont construites et hiérarchisées. Nous avons aussi détaillé les modèles de processus d'objets pour la machine et la station dans le but d'illustrer comment nous pouvons implémenter la dynamique du système.

Le chapitre V est consacré à l'implémentation des modèles de simulation concurrente. Nous avons commencé par les concepts de base de la programmation concurrente: la notion de processus et les relations entre les processus en terme informatique. Ensuite, nous avons présenté trois langages ou outils basés sur le langage de programmation C++ qui permettent d'implémenter les modèles de simulation. Enfin nous en avons choisi un pour illustrer comment nous avons implémenté les classes d'objets des systèmes de production.

La conclusion générale fournit un bilan des travaux que nous avons menés ainsi que les travaux et objectifs futurs concernant la simulation orientée-objets.

Chapitre I Systèmes de Production et Gestion de Production

I Systèmes de Production

La *production* est une opération de transformation qui convertit des matières premières et/ou des composants que l'on peut qualifier de "bruts" (au sens le plus général) en produits finis plus élaborés et de valeur économique plus élevée.

Un *système de production* est un ensemble de ressources qui permettent cette transformation. Dans cet ensemble, on distingue essentiellement quatre types de *ressources*: des équipements (machines, outils, moyens de transport, moyens informatiques, ...), des moyens humains qui permettent le bon déroulement du processus de transformation, des produits à différentes étapes de fabrication (matières premières, produits semi-finis, produits finis, ...), des entrepôts de matières ou des aires de stockage.

En ce qui concerne les équipements de production, on distingue trois sous-types: les machines de production, permettant d'effectuer des opérations de transformation, les machines de manutention, permettant de transporter des pièces dans l'atelier (robots, chariots mobiles, tapis roulant, transtockeurs, ...), et les machines de contrôle de qualité. Les deux dernières peuvent être considérées comme des machines de production spéciales ou fictives.

Un *produit fini* est généralement obtenu par assemblage de plusieurs composants. Un *composant* est à son tour obtenu par assemblage d'autres pièces ou par une succession de transformations de matières premières. Les matières premières, les composants (produits semi-finis) ainsi que les produits finis sont des *articles*, qui se distinguent les uns des autres par leur état de transformation. Un article qui subit une opération de transformation devient un article à l'état différent. Un article (en cours de transformation) est souvent appelé une *"pièce"* dans la production.

La description complète du processus de fabrication d'un produit fini est donnée par une arborescence des composants, connue sous le nom de *nomenclature*, et la description du processus de chaque composant est donnée par sa *gamme de fabrication*. Une gamme de fabrication est l'ensemble des opérations qui conduisent à l'achèvement d'un composant ou d'un produit dans le cadre du système de production.

Face à la complexité du processus de fabrication, les systèmes de production sont organisés et gérés en fonction des demandes et des ressources disponibles. On trouve dans la littérature de nombreuses typologies des systèmes de production. Nous considérons que les deux typologies intéressantes proposées par Giard [GIARD 88] sont les plus naturelles et les plus synthétiques bien qu'il y ait beaucoup de proposition dans les littératures.

La première typologie est basée sur le fait qu'un système de production peut produire soit pour réapprovisionner des stocks (production prévisionnelle) soit pour satisfaire une demande (production à la demande). Une production est prévisionnelle lorsqu'elle est déclenchée par un état de stock. Un système de production produit à la demande lorsque la production est déclenchée par les demandes fermes des clients (voir section II.1).

La production prévisionnelle conduit à des modèles déterministes ou stochastiques de gestion de stocks. Parmi ces modèles, on trouve en particulier les fameux modèle de quantités économiques qui assurent un compromis optimal entre les coûts de stockages et les coûts d'obtention de produits ou les coûts d'approvisionnement [TERSINE 88].

On trouve également la combinaison de ces deux modes de fabrication dans des systèmes qui comportent à la fois la fabrication des composants et l'assemblage. On produit souvent les composants de manière prévisionnelle et l'assemblage se fait à la demande des clients. Ceci permet de fournir une grande variété de produits assemblés à partir d'un nombre limité de types de composants.

La deuxième typologie est liée au mode d'organisation de la production ([MULKENS 93], [JAVEL 93]). On peut distinguer les quatre modes d'organisation suivants:

- *Organisation en ligne de fabrication.* Dans les systèmes organisés en ligne de fabrication, les équipements sont agencés de telle façon que les produits sont fabriqués en passant successivement et dans le même ordre par les postes de travail de la ligne. Ce mode d'organisation nécessite la spécialisation des équipements. Il repose sur la parfaite standardisation des gammes de fabrication et la régularité de la circulation du flux de matières. Ce mode d'organisation permet une très bonne utilisation des équipements. Les temps d'attente des pièces pour la fabrication sont faibles. Un autre avantage de ce mode d'organisation est la simplicité de sa gestion. Il n'est pratiquement pas nécessaire d'ordonnancer la production. La rigidité de ce mode est justifiée lorsque la quantité de produits fabriqués est suffisamment grande. C'est le cas dans les productions de masse.

- *Organisation en ateliers spécialisés (cellules flexibles)*. Dans ce type d'organisation, on réunit en un même lieu les équipements qui assurent une même fonction technique. Ce mode d'organisation est la conséquence de la diversité des articles à transformer. Chacun de ces ateliers fait l'objet d'une production limitée. Dans ce mode d'organisation, les machines ne sont généralement pas spécifiques et peuvent effectuer plusieurs types d'opérations. L'avantage de ce mode d'organisation est la flexibilité du système. Le principal inconvénient est l'inefficacité de la fabrication par rapport à une organisation en ligne. En effet, le coût et le temps de manutention des articles entre les ateliers et les machines sont souvent importants. De plus, la diversité des gammes de fabrication (produits) dans chaque atelier pose un problème d'ordonnancement difficile. Les temps d'attente des articles pour la fabrication sont souvent plus importants que dans l'organisation en ligne. En même temps, les machines ne sont pas aussi bien utilisées que dans l'organisation en ligne de fabrication.

- *Organisation en production unitaire*. Il s'agit de la réalisation de quelques produits sur des périodes assez longues, par exemple la fabrication des prototypes. Il est nécessaire de planifier la succession des différentes tâches. Un diagramme de type PERT, par exemple, est souvent utilisé pour la planification des tâches et de son suivi.

- *Industrie de processus ou production continue*. Les produits fabriqués subissent des transformations pratiquement continues. On rencontre, par exemple, ce mode d'organisation dans l'industrie chimique ou sidérurgique.

Dans ce travail, nous nous intéressons aux systèmes de production fonctionnant en prévisionnel et organisés en ateliers spécialisés. C'est le cas le plus général dans l'industrie.

II Gestion de Production

L'objectif d'un système de production est de fournir aux clients des produits de bonne qualité, à un prix compétitif et en respectant les délais. L'entreprise doit chercher en permanence à améliorer ses produits et ses délais de livraison, elle doit en outre diminuer constamment le coût de production.

Le système de gestion a pour rôle d'assurer en permanence la bonne utilisation de l'ensemble des moyens de production. Pour bien gérer la production, les décisions prises par le système de gestion doivent être basées sur des informations actualisées (état du système de productiion, ressources disponibles, situation des clients et des fournisseurs). Un bon système de gestion est

caractérisé par sa capacité d'acquisition des informations nécessaires et par sa capacité de prise de décisions en cas nécessaire.

Si l'on suit le mouvement des matières dans un système de production, on peut décomposer son fonctionnement en trois étapes:

- Etape de réapprovisionnement qui consiste à commander aux fournisseurs les matières premières et les composants requis pour la fabrication.

- Etape de fabrication qui transforme les matières premières et les composants en produits finis.

- Etape de distribution qui assure la livraison des produits finis aux clients.

Le mauvais fonctionnement d'une étape quelconque perturbe le fonctionnement de l'ensemble. Le système de gestion doit coordonner le fonctionnement de ces étapes.

La figure suivante (figure 1-1) schématise les interactions des sous-systèmes dans un système de production. En terme d'automatique, c'est une gestion avec retour d'information en boucle fermée.

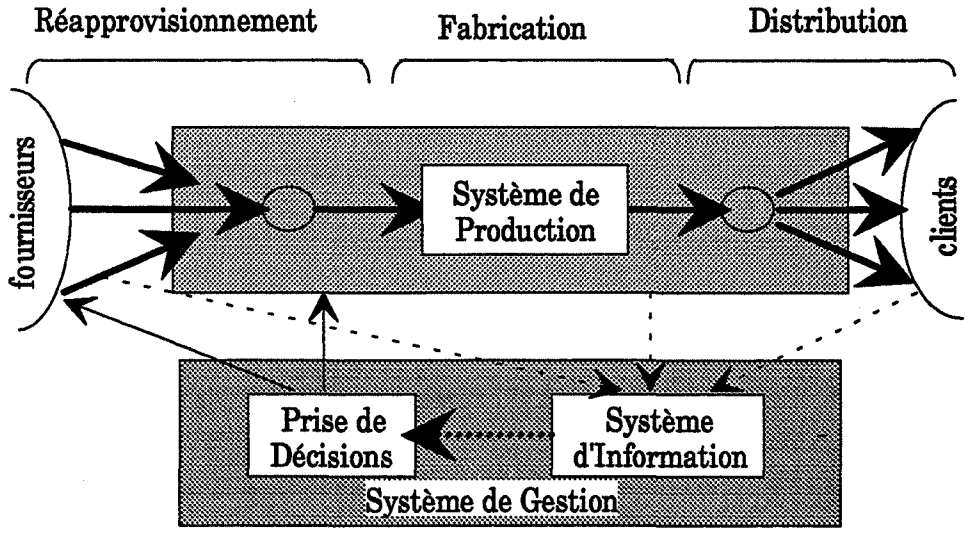


Figure 1-1 Interactions des Sous-Systèmes dans l'Entreprise

Le système d'information collecte les informations sur la capacité actuelle de fabrication, les fournisseurs, les demandes, les prévisions, etc. Il permet également d'évaluer les décisions prises par le système de gestion et de suivre l'évolution du marché et de la technologie. Il met

le système de gestion au courant de tout changement du système de production et du monde extérieur ce qui permet une adaptation rapide des décisions.

Les décisions sont prises en fonction des informations collectées par le système d'information. Pour un système de production, il y a en permanence de nombreuses décisions à prendre. De plus, il existe souvent des aléas qui perturbent le fonctionnement du système. Cette instabilité de l'état du système rend naturellement la prise de décision plus difficile.

Une autre difficulté de la gestion est due au fait que les objectifs ne sont pas clairement définis. Souvent, on a une liste d'objectifs souhaités qui peuvent être contradictoires, par exemple:

- minimiser les en-cours et les stocks;
- livrer les produits aux clients dans le plus court délai;
- utiliser au mieux la capacité des moyens de production et les personnels disponibles;
- minimiser les heures supplémentaires;
- minimiser le coût de production;
- etc.

Pour atteindre ces objectifs, il faut tout d'abord abandonner la mise en oeuvre de la gestion de production technique par technique, au profit d'activités concomitantes et surtout cohérentes. Donnons un exemple pour le prouver.

Une technique de gestion de stocks a pour objectif le calcul de quantités économiques de stockage pour chaque produit référencé dans l'entreprise. Ce calcul s'inspire souvent de la formule de Wilson, qui détermine cette quantité par un compromis entre le coût de stockage et le coût d'approvisionnement. Le coût d'approvisionnement est d'autant faible que la quantité approvisionnée (donc en stock) est élevée (coût d'approvisionnement, coût de réglage d'une machine, etc.).

Une autre technique: le contrôle de qualité a souvent pour objectif la mesure statistique de la qualité des produits. Les indicateurs de qualité issus de ces calculs préconisent d'agir sur les procédés de fabrication ou sur les moyens de production à l'origine de la non-conformité des produits. La mesure statistique d'une caractéristique d'un produit s'effectue sur plusieurs échantillons. Ces échantillons sont extraits du stock du produit, dont la quantité est déterminée par ailleurs, comme nous l'avons décrit dans la technique de gestion de stocks. Remarquons que la mesure statistique est d'autant plus représentative de la qualité du produit que la quantité en stock est faible (représentativité des échantillons par rapport à la population en stock).

Dans l'application de la première technique, on tend à augmenter le stock, alors que dans l'application de la seconde, on souhaite qu'il diminue. La contradiction qui apparaît ici, montre la dépendance entre deux problèmes de la production: adopter une politique de qualité efficace oblige à une gestion différente de stocks. Le système de gestion doit aboutir à un bon compromis entre ces objectifs contradictoires.

II.1 Classification des Décisions

Afin de réduire la difficulté de la prise de décisions, on adopte une approche progressive. Les décisions sont souvent prises en trois étapes successives caractérisées par l'horizon sur lequel les décisions s'appliquent [XIE 89]:

- *Les décisions stratégiques* qui concernent la politique générale de l'entreprise à long terme (horizon de plus de deux ans, en général). A partir de l'analyse de la tendance du marché, de la prévision de l'évolution des technologies et de l'analyse de la concurrence, les décisions stratégiques cherchent à faire évoluer l'ensemble des produits et à ajuster le mode d'organisation et la capacité de la production aux besoins du marché. Les décisions stratégiques se traduisent par:

  - investissement en équipements, recrutement et formation du personnel;
  - arrêt de fabrication de certains produits et lancement en fabrication de nouveaux produits;
  - conception de nouveaux produits;
  - adaptation à de nouveaux modes de production, par exemple Juste-A-Temps;
  - programme publicitaire;
  - etc.

- *Les décisions tactiques* qui correspondent à un ensemble de décisions à moyen terme (horizon variant entre 3 mois et 2 ans, en général). La capacité de production a été fixée par le niveau supérieur (décisions stratégiques). A partir du carnet de commandes fermes des clients et de la prévision des demandes, les décisions tactiques définissent un plan de fabrication. Elles se traduisent par un calendrier de production (programme de production).

- *Les décisions opérationnelles* qui contrôlent le déroulement quotidien du processus de production dans le respect des décisions tactiques. Les décisions opérationnelles assurent l'ordonnancement des opérations sur les machines, l'affectation des opérateurs aux machines, la livraison des produits finis aux clients, etc. Elles tiennent compte de tous les détails du fonctionnement du système de production.

La figure suivante (figure 1-2) schématise les interactions entre ces trois niveaux de décisions. A chaque classe de décision correspond un horizon différent. Les informations nécessaires sont d'autant plus agrégées que l'horizon éloigne. De plus, les décisions prises concernent d'autant plus d'éléments que le niveau de décision est élevé.

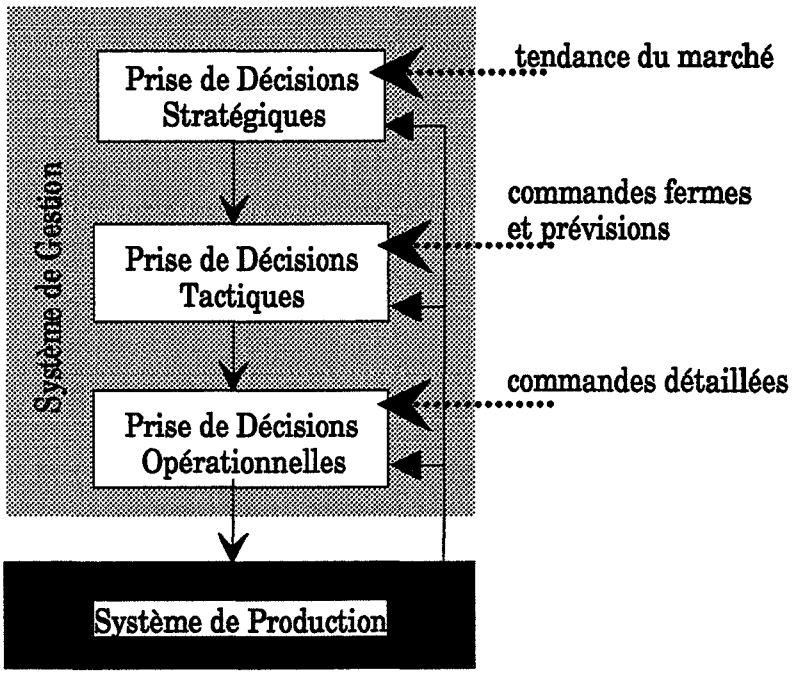


Figure 1-2 Interactions entre les Décisions en Gestion de Production

Au niveau des décisions stratégiques, seules les prévisions sur l'évaluation du marché et de la technologie sont prises en compte. Les décisions stratégiques ont des effets globaux et profonds sur le fonctionnement du système. C'est uniquement à ce niveau que l'on effectue les prises des décisions importantes sur l'évolution du système physique (organisation du système).

Au niveau des décisions tactiques, on considère les commandes fermes, les prévisions de demande regroupées en familles de produits, et les capacités agrégées des sous-systèmes de production. Les décisions sont les productions globales sur des périodes élémentaires relativement importantes (le mois ou le trimestre, par exemple).

Les décisions opérationnelles concernent le détail du fonctionnement du système en temps réel.

Toute décision prise à un niveau devient une contrainte pour les décisions du niveau immédiatement inférieur. Un retour d'information vers les niveaux supérieurs est nécessaire afin de prendre en compte les changements d'états résultant de l'application des décisions (suivi).

Un problème important est de définir l'information à fournir à chaque niveau, sa forme, sa périodicité, son niveau d'agrégation, etc.

Dans ce mémoire, nous nous intéressons aux décisions tactiques et opérationnelles, c'est-à-dire à la planification de la production et à son ordonnancement.

II.2 Fonctions de Gestion

Nous étudierons ici les différentes fonctions de gestion à court et moyen terme qui correspondent aux décisions tactiques et opérationnelles décrites dans le paragraphe précédent. La prise de ces décisions peut, dans la pratique industrielle, être divisée en trois phases: planification, programmation et exécution.

II.2.1 Phase de Planification

Dans la phase de planification, il s'agit d'établir un *plan directeur de production*. On a un carnet de commandes fermes et la prévision des demandes futures. Les décisions portent essentiellement sur la production de familles de produits finis sur un intervalle de temps appelé horizon de planification. L'objectif est de satisfaire la demande en minimisant les stocks, les retards, le coût de production, etc. Cette phase correspond au niveau des décisions tactiques. En planification de la production, on distingue deux approches fondamentalement différentes: l'approche globale et l'approche hiérarchisée [XIE 89].

L'*approche globale* utilise un modèle monolithique qui décrit l'ensemble du problème de planification de manière détaillée. Elle fournit toutes les décisions sur l'horizon complet. Mais l'utilisation de cette approche, basée essentiellement sur la programmation linéaire en variables mixtes, conduit à des programmes de grandes tailles et des temps de calcul exponentiels [GERSHWIN 89]. A notre connaissance, il n'y a pas de méthodes qui puissent fournir en un temps raisonnable le plan de production d'un système réel par une approche globale. D'autre part, beaucoup de données sont prévisionnelles. La grande quantité de données nécessite un système de prévision complexe. Il est sans intérêt de considérer les données de façon détaillée sur l'horizon complet.

L'*approche hiérarchique* est proposée par de nombreux auteurs. On trouve l'état de l'art dans Hax et Meal [HAX et al. 75] et Giard [GIARD 88]. Cette approche décompose le problème de planification en sous-problèmes. Chaque sous-problème est lié à un niveau dans une hiérarchie. A chaque niveau, les entités sont des agrégats des entités du niveau immédiatement inférieur.

L'horizon de planification décroît en descendant dans la hiérarchie. Les décisions correspondant à ces agrégats sont désagrégées au niveau immédiatement inférieur.

Les travaux de Hax et Meal sont basés sur une agrégation logique des produits: *articles*, *familles*, *types*, et ont conduit à l'élaboration d'une planification en trois étapes:

- La première étape détermine un plan de production des types de produits sur l'horizon complet de la production. Le modèle utilisé est un modèle linéaire, qui prend en compte la capacité du système, les heures de travail régulières et supplémentaires, le coût de stockage et le coût de fabrication, pour minimiser un coût de gestion et satisfaire la demande de chaque type de produits sur la période considérée.

- La deuxième étape correspond à une désagrégation de la programmation par type de produits en une programmation par famille de produits. Cette désagrégation est faite uniquement pour la première période de l'horizon. Pour la période suivante, la programmation sera faite en tenant compte des résultats réels du déroulement de la première période afin que des ajustements puissent être effectués. L'objectif retenu est de minimiser la somme des coûts de lancement de la production des familles qui constituent ce type.

- La troisième étape est une désagrégation de la programmation par famille de références en programmation par article de références.

Cette approche hiérarchisée présente l'avantage de ne comporter qu'un type de prévision, celui de la demande des types de produits et de tenir compte du déroulement du processus de fabrication en appliquant une programmation période par période.

II.2.2 Phase de Programmation

La phase de programmation consiste à calculer les besoins en composants et les besoins en capacité en fonction du plan directeur de production. La *planification des besoins en composants* (Material Requirement Planning ou M.R.P.) détermine quels composants et matières premières il faut approvisionner ou fabriquer, en quelles quantités et à quelles dates. Ces calculs sont basés sur l'élaboration des gammes de fabrication (ou nomenclatures) des produits finis. A l'issue de la planification des besoins en composants, la *planification des besoins en capacité* quantifie les ressources nécessaires de fabrication. Ces informations permettent de déterminer le nombre de postes de travail à utiliser, la répartition des opérateurs et le nombre d'heures supplémentaires. Cette phase est largement étudiée dans la littérature et

de nombreux logiciels de type MRP sont implantés dans les entreprises [ORLICKY 75]. Elle se situe également au niveau des décisions tactiques.

II.2.3 Phase d'Exécution

Dans cette phase, on affronte les détails du déroulement de la production. Dans les activités de production proprement dite, la fonction d'*ordonnancement d'atelier* a pour objectif d'établir un agenda de production pour chaque machine afin de fabriquer les pièces en temps voulu. Cet agenda est sujet aux modifications dues aux aléas du système de production. Une fonction de *suivi de production* a pour rôle l'acquisition des informations sur le déroulement de la production, ce qui permet la modification des agendas de production en cas de perturbations. Cette dernière phase se situe au niveau des décisions opérationnelles.

L'ordonnancement de la production a pour rôle d'organiser l'exécution des opérations sur les machines dans chaque atelier. Il doit réagir à tout changement non prévu tel que les pannes des machines, les ruptures (manques) de matières premières, l'absence de personnel, etc.

Les méthodes de résolution des problèmes d'ordonnancement puisent dans toutes les techniques de l'optimisation combinatoire (programmation linéaire ou dynamique, procédures par séparation et évaluation, théorie des graphes, etc.). Ces méthodes garantissent en général l'optimalité de la solution fournie. Mais les algorithmes dont la complexité n'est pas polynomiale ne peuvent pas être utilisés pour des problèmes de grande taille, d'où la nécessité de construire des méthodes de résolution approchée, efficaces pour ces problèmes souvent NP-difficiles [CARLIER et al. 88]. Différentes approches de résolution sont proposées dans la littérature; on distingue les cinq méthodes suivantes: par construction progressive, par voisinage, par décomposition, par relaxation et enfin celles liées à l'intelligence artificielle. On trouve l'état de l'art dans l'article de GOTHA [GOTHA 93].

Les méthodes par construction progressive sont des méthodes itératives où, à chaque itération, on complète une solution partielle. Contrairement aux méthodes par construction qui travaillent sur des solutions partielles, *les méthodes par voisinage* travaillent sur des solutions complètes. Chaque itération de la méthode par voisinage ayant pour objectif (pas toujours atteint) de passer d'une solution complète à une autre solution complète meilleure par rapport au critère considéré.

En ce qui concerne *les méthodes par décomposition*, il existe de nombreuses façons de les construire:

- la décomposition *hiérarchique* [ERSCHLER et al. 85] consiste à décomposer les problèmes en plusieurs niveaux, par exemple un niveau supérieur où on travaille sur des données agrégées et où on décide d'un nombre de paramètres globaux et un niveau inférieur détaillé où les décisions prises sur ces paramètres globaux deviennent des contraintes pour le niveau inférieur, ce qui limite le domaine des solutions à explorer.

- la décomposition *structurelle* [ROY 70] utilise la modélisation du problème et ses grandes familles d'inconnues; elle considère que les moyens sont illimités et résout le problème temporel. Les dates étant fixées, on cherche la meilleure affectation possible des moyens, puis on revient au problème temporel en ajoutant de nouvelles contraintes liées à l'utilisation de ces moyens, etc.

- la décomposition de *l'ensemble des solutions du problème* conduit, si l'on veut obtenir la solution optimale, à des procédures par séparation et évaluation. Toutefois, ces méthodes exactes peuvent être transformées en heuristiques de différentes façons, par exemple, dans le cas du *recuit simulé*, en les arrêtant lorsque pendant Q itérations on n'a pas strictement amélioré la meilleure solution trouvée ou encore, lors des séparations, en n'essayant pas toutes les valeurs possibles pour l'inconnue sur laquelle on affecte la séparation, mais seulement un échantillon de valeurs possibles.

- la décomposition *temporelle* [PORTMANN 87], dans le cas des ordonnancements dynamiques où les travaux arrivent à des dates différentes et dispersées dans le temps, résout lors de la première itération un problème réduit en ignorant les travaux qui arrivent au-delà d'une date choisie, puis retient définitivement cet ordonnancement partiel jusqu'à une autre date déterminée. A l'itération suivante, on décalera ces deux dates de manière à construire la portion suivante de l'ordonnancement.

- la décomposition *spatiale* [PORTMANN 87] décompose l'atelier en sous-ateliers avec le moins possible de déplacements de produits entre les sous-ateliers grâce à des outils de *technologie de groupe* et construit une méthode itérative dans laquelle chaque itération ordonnance un sous-atelier.

Les méthodes par modification des contraintes essaient de changer le modèle des problèmes que l'on a à résoudre. Ce peut être, par exemple, de transformer un flow-shop normal en un flow-shop de permutation de manière à avoir moins de solutions à explorer, mais on trouve surtout ici toutes les méthodes dites de "relaxation": relaxation de contraintes d'intégrité, relaxation lagrangienne, relaxation "surrogate", etc.

Les méthodes liées à l'intelligence artificielle utilisent des techniques de représentation des connaissances et de résolution de problèmes issues de l'intelligence artificielle. De nombreux travaux ont abordé les problèmes d'ordonnancement à l'aide de ces outils. L'utilisation des techniques de l'intelligence artificielle, en particulier les systèmes experts d'ordonnancement d'atelier, permet de manipuler avec souplesse des ensembles d'heuristiques plus nombreux et plus précis [BEL 88]. Certains systèmes ont été implantés avec succès dans l'industrie [BEL et al. 88] tels que ISIS (Intelligent Scheduling and Information Systems), OPIS (Opportunistic Intelligent Schedules), SOJA (Système d'Ordonnancement Journalier d'Atelier), PLANNEX et OPAL, etc. L'intégration de la simulation dans la résolution de problèmes d'ordonnancement [JAIN et al. 90] [YE et al. 92a] permet d'ordonnancer en temps réel ("on-line") un atelier en améliorant les performances des méthodes d'ordonnancement.

En conclusion, l'ordonnancement est un domaine où les problèmes à résoudre sont difficiles. La difficulté est à la fois théorique et pratique. Théorique parce que la plupart des énoncés correspondent à des problèmes combinatoires de complexité exponentielle (NP-difficiles). Pratique parce que les particularités de chaque atelier font que la solution obtenue à grand frais n'est pas toujours utilisable: on a souvent négligé des aspects technologiques ou humains difficilement modélisables par les méthodes classiques de recherche opérationnelle. Les problèmes réels sont donc mal résolus et paraissent a priori très complexes; on peut cependant souvent trouver des méthodes de résolution très satisfaisantes. Ces problèmes sont distincts les uns des autres et ne peuvent pas être traités efficacement à l'aide d'un outil standard. Une partie au moins de la recherche théorique est proche des cas réels. Les outils théoriques permettent à l'heure actuelle de résoudre certains problèmes de grande taille (par exemple le job-shop). Les bonnes heuristiques, suffisantes le plus souvent, sont des sous-produits d'études théoriques fines [GOTHA 93].

III Méthodes de Gestion de Production

Les méthodes de gestion de production ont à l'origine privilégié une logique de gestion plutôt qu'une autre. Parmi les plus célèbres outils de gestion de production assistée par ordinateur (GPAO), citons: "gérer par une planification" pour la méthode M.R.P. II, "gérer en juste-à-temps" pour la méthode Kanban, et "gérer par les moyens de production signifiés pour leurs goulots d'étranglement" pour la méthode O.P.T.

Rien ne prouve cependant qu'il faille se limiter à ces logiques. Plus généralement, un reproche que l'on peut faire à ces méthodes, est de figer les règles de gestion, a fortiori lorsqu'elles sont décrites dans un logiciel de GPAO, qui sont difficiles à adapter à chaque entreprise, voire à appliquer, compte tenu des différents types de production rencontrés. Nous nous intéressons

dans ces travaux, pour l'essentiel, aux outils logiciels ou modèles qui permettent la mise en oeuvre des méthodes, et à cette occasion, à la manière de ne plus figer les règles de gestion au sein de ces outils. Nous nous préoccupons de la gestion de production sous cet angle, et pensons que la gestion de production n'est donc pas qu'un problème de méthode. Nous présentons dans la suite les méthodes les plus connues.

III.1 La Méthode M.R.P.

La méthode de gestion de production M.R.P. (Manufacturing Resource Planning ou Material Requirement Planning) [ORLICKY 75] est traditionnellement associée à la logique de gestion par planification. Elle trouve son origine sur la base des deux analyses suivantes :

1) Les produits appartenant au flux matière circulant dans l'entreprise, sont l'objet de deux types de besoins [BELT et al. 86]. D'une part, les *besoins indépendants* de la volonté directe de l'entreprise, c'est-à-dire les besoins qui s'expriment d'une façon externe et aléatoire par rapport à l'entreprise: ce sont les commandes que l'on n'est jamais sûr d'obtenir ou encore les besoins de pièces de rechange pour l'après-vente. D'autre part, les *besoins dépendants* de l'entreprise elle-même, c'est-à-dire les besoins qui peuvent être exprimés d'une façon déterministe à partir de ceux qui ne peuvent être qu'estimés, concernant tous les composants d'assemblage ou de fabrication. Les besoins indépendants ne peuvent donc qu'être exprimés (commandes fermes) ou prévus (demandes et commandes prévisionnelles), par contre, les besoins dépendants sont calculés grâce aux nomenclatures des besoins indépendants qui décrivent la dépendance structurelle de fabrication des produits.

2) La planification de tous ces besoins est nécessaire. Une planification a pour but de décrire, pour une échelle temporelle définie par un horizon de planification (i.e. une limite de validité) et une période de réactualisation, un *plan de production*. Un plan de production répond entre autres aux questions :

- que produire ?
- quand produire ?
- quelle quantité produire ?

Un plan de production est une "photographie" de la production pour un horizon déterminé et regroupe de ce fait des informations de nature à expliquer l'évolution de la production. On trouve donc parfois dans un plan de production des informations sur les ventes, les stocks et les activités de production.

La planification vise à dégager une capacité d'adaptation accrue de l'entreprise: répondre aux besoins des clients en temps voulu est en effet obtenu ici par *anticipation*. Pour cette raison, la méthode M.R.P. II propose le schéma de gestion de la figure suivante (figure 1-3).

La logique de gestion par planification apparaît dans la méthode M.R.P. II, au travers de l'enchaînement de plans de production, où figurent des informations cohérentes d'un niveau à un autre. Nous proposons d'expliquer cet enchaînement pour comprendre ce qu'est un système de planification.

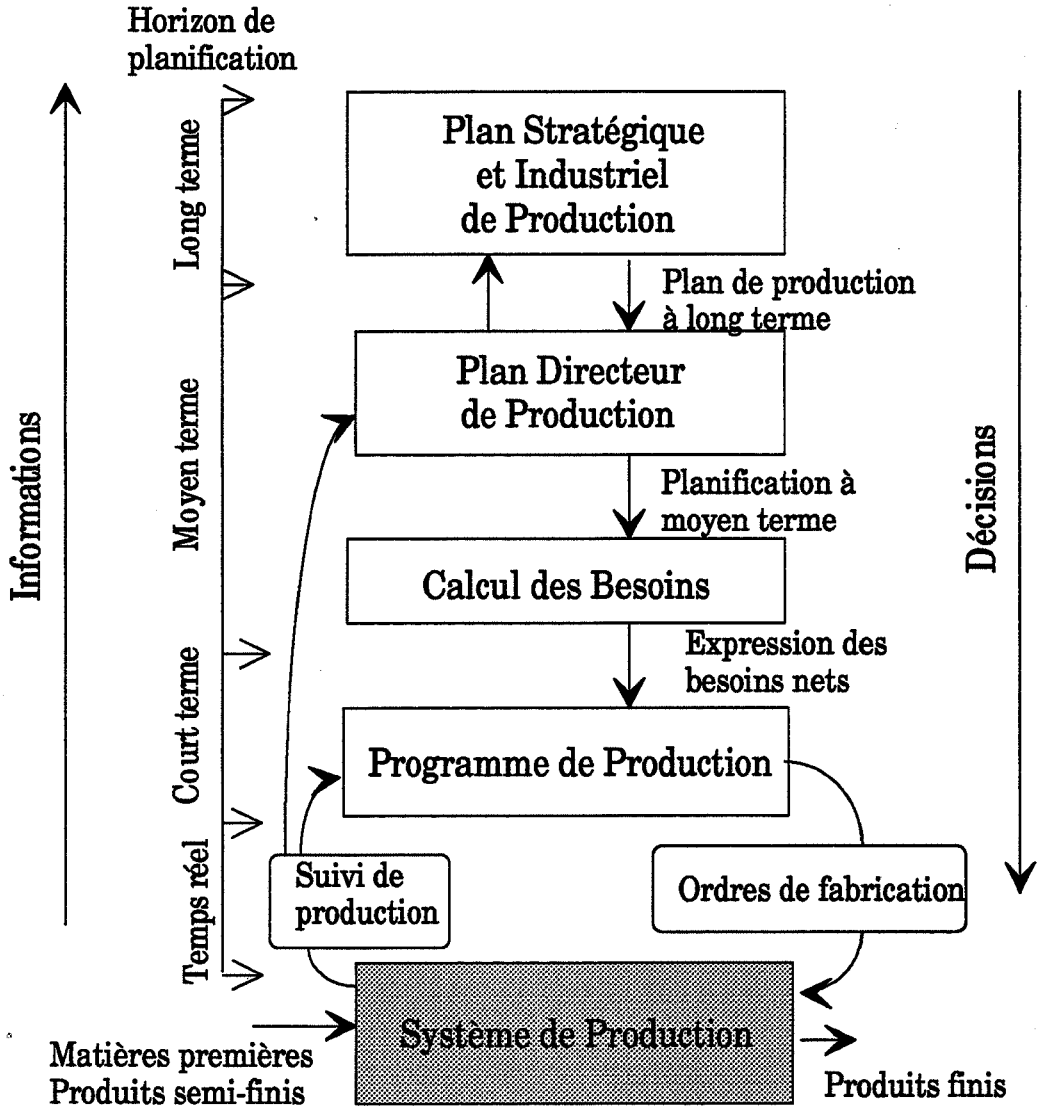


Figure 1-3 Méthode M.R.P. II

III.1.1 Plan Stratégique et Industriel de Production

Le plan stratégique et industriel de production (PSIP) exprime les ventes connues et espérées des familles de produits, ainsi que la production et les stocks, réels et prévisionnels de ces familles. Son utilité est justifiée par le fait que les prévisions de vente par familles de produits sont plus faciles à établir que celles sur les produits eux-mêmes. On se base essentiellement sur une analyse de l'historique du passé: on cherche à définir le modèle des commandes passées le plus proche de la réalité. Ce modèle va nous servir à faire des prévisions sur les demandes futures. Ce plan confronté aux capacités de production, aux capacités de financement et aux capacités techniques permet de déterminer, en raisonnant sur les familles de produits, le niveau de production mensuel acceptable fixant l'objectif stratégique de l'entreprise. Son horizon de planification est généralement l'année et est découpé par périodes de réactualisation d'un mois.

III.1.2 Plan Directeur de Production

A partir des prévisions de tendances du marché et de comportement du système de production et de la stratégie à suivre, le plan directeur de production (PDP) traduit les objectifs de ces prévisions exprimées en familles de produits. En effet, on ne fabrique pas une famille de produits mais un produit, et on n'approvisionne pas des familles de composants ou des familles de matières premières, d'où la nécessité d'une traduction de cette stratégie. Les paramètres de prévisions sont détaillés pour chaque produit et ne sont plus relatifs à des familles de produits comme c'était le cas pour le PDP. Son horizon de planification est en général de quelques mois et est découpé par périodes de réactualisation d'une semaine. Notons que cet horizon de planification est dépendant des types de produits fabriqués par l'entreprise. On préfère ainsi fixer cet horizon en fonction du cycle technique de production. Le cycle technique de production est le temps entre la date de mise en fabrication d'un produit et sa date d'entrée en stock ou sa date de livraison. Il est calculé par la somme des temps opératoires sur la gamme de fabrication (à travers sa nomenclature) et inter-opératoires (temps de transport et temps d'attente) du produit. Contrairement au PSIP, le PDP est exécutoire, en ce sens qu'il fixe le niveau d'activité de la production de l'entreprise.

III.1.3 Calcul des Besoins

Le but est de déterminer, pour une période donnée, les besoins en articles à fabriquer (composants ou produits finis) ou à approvisionner (composants ou matières premières). Ces besoins expriment les fabrications (ordres de fabrication) et les approvisionnements (ordres d'achat) à réaliser en quantité et date. Il y a deux types de calcul des besoins: *calcul des besoins bruts* et *calcul des besoins nets.* Le calcul des besoins bruts est réalisé à partir des nomenclatures des produits. Une nomenclature donne les produits dont on doit disposer en

stock et les composants dont il faut disposer (achetés ou fabriqués). Les dates des besoins exprimées par le calcul des besoins bruts sont comparées aux stocks prévisionnels. Ceci permet de déduire des ordres de fabrication (O.F.) ou des ordres d'achat (O.A.).

La quantité des besoins est une quantité économique et/ou technique, qui satisfait des contraintes de coût de production et/ou des contraintes techniques de fabrication, en donnant lieu à des regroupements en *lots de fabrication*. La date de mise en fabrication est calculée en fonction de la date de mise à disposition des quantités, en tenant compte des délais de fabrication. Cette date de mise en fabrication est en général calculée au plus tard (date limite). Toute quantité tient aussi compte des stocks résiduels de fabrication d'un produit.

III.1.4 Programme de Production

Le résultat du calcul des besoins nets est un ensemble d'ordres de fabrication. Chaque ordre est caractérisé par une quantité et une date. Ces O.F. (prévisionnels) ont été élaborés sans tenir compte de la capacité réelle du système de production. On parle alors d'ordonnancement à capacité infinie.

Le programme de production a pour but d'ajuster les capacités aux charges. La capacité d'un moyen de production mesure la production maximale en unités de produits par unités de temps. La charge d'un moyen de production mesure elle, l'utilisation sur une période d'un nombre d'unités de capacité. En outre, l'ajustement a pour but d'assurer que :

- la charge est inférieure à la capacité. C'est une contrainte matérielle évidente.

- la charge tend vers la capacité. C'est une contrainte économique importante, car elle permet de rentabiliser l'usage des moyens de production.

La méthode M.R.P. II traite grossièrement les problèmes d'ajustement des capacités aux charges avant la constitution du programme de production. Cet ajustement, qui peut dégager des capacités suplémentaires si nécessaire et si possible, est fait par groupe de moyens de production, et non par moyen de production.

Le programme de production établit la transition dans la méthode M.R.P. entre la planification et l'ordonnancement des tâches de fabrication. Son horizon est d'environ une, voire deux semaines. Le résultat du calcul des besoins qui détermine un ordre de réalisation des O.F. est qualifié d'ordonnancement à capacité infinie. L'objet essentiel du programme de fabrication est de substituer à cet ordre, un ordre de passage des tâches associées aux O.F.. Une tâche est le travail résultant d'une opération d'une gamme de fabrication associée à un O.F.. A une tâche

correspond donc une charge détaillée pour chaque moyen de production utilisé par l'opération. Le résultat du calcul de l'ordre est appelé dans ce cas, un ordonnancement à capacité finie, puisqu'il est tenu compte des limites de la capacité de chaque moyen.

La constitution du programme de production nécessite également le choix des O.F. à réaliser. Ce choix est appelé le *lancement*. Il dépend principalement du suivi de production, c'est-à-dire de l'état de la production à tout instant connu comme, par exemple, la disponibilité des moyens de production, l'état d'avancement des O.F. déjà lancés, etc.

Ainsi, la constitution du programme de production dans la méthode M.R.P. II a un caractère plus heuristique que méthodique. Les causes majeures en sont:

- la mauvaise qualité du système d'information mis en place: le volume, la localisation, la cohérence et la communication de l'information rendent difficile la mise à jour du programme de production.

- le caractère évolutif du programme de production. Les aléas de fabrication provoquent sans cesse sa réactualisation. Cependant le temps imparti pour calculer à nouveau ce programme empêche souvent sa mise à jour. La technique de l'ordonnancement, lorsqu'elle existe, a donc ici un rôle important à jouer.

III.1.5 Conclusion sur la Méthode M.R.P.

La méthode M.R.P. II peut être qualifiée de méthode de planification. En effet, elle ne traite que les niveaux long et moyen termes et ignore la gestion du court terme, alors qu'il est difficile d'appliquer des stratégies de gestion, même très élaborées, à un système si on n'étudie pas l'application de ces stratégies et les réactions du système dans le court terme. Ceci ne veut pas dire que M.R.P. est à abandonner, d'autant plus que la majorité des logiciels de gestion de production actuels sont basés sur cette méthode, mais à compléter pour couvrir efficacement le court terme. En tout état de cause, il est difficile d'ignorer la spécificité de la production dans l'entreprise, lorsque l'on cherche à la gérer. Une approche possible pour appréhender est de considérer certaines propriétés de la production et en tenir en compte dans la méthode de gestion. La méthode juste-à-temps proposée dans la section suivante suit cette approche.

III.2 La Méthode Juste-A-Temps (J.A.T.) et la Méthode Kanban

III.2.1 La Méthode Juste-A-Temps

Le juste-à-temps est une logique de production. Utiliser le juste-à-temps c'est acheter ou produire juste ce qu'il faut à la date exacte du besoin. Le but principal du juste-à-temps est la réduction des coûts et non pas l'optimisation de l'utilisation des moyens de production. La réduction des coûts est obtenue par la réduction des stocks elle même résultat d'une bonne synchronisation entre la demande et la production. En effet, la production d'un produit ou d'un composant est déclenchée par la demande (production à flux tirés) et non plus à partir de prévisions (production à flux poussés) comme c'était le cas pour la méthode M.R.P.

Le concept juste-à-temps est à l'origine du concept de "flux tendu". Il est lié très étroitement aux démarches de "qualité totale" et d'"amélioration permanente". Le concept de flux tendu résulte des concepts des cinq zéros (figure 1-4 [MILLE 93]): zéro défaut, zéro panne, zéro campagne de production, zéro stock et zéro délai. Les deux premiers zéros essaient d'éliminer tous les aléas de production et le troisième de supprimer tous les processus de production en campagnes (secondaires). Ce n'est qu'après avoir atteint ces trois zéros que le juste-à-temps pourra être mis en oeuvre pour conduire les deux derniers zéros: zéro gaspillage et zéro stock.

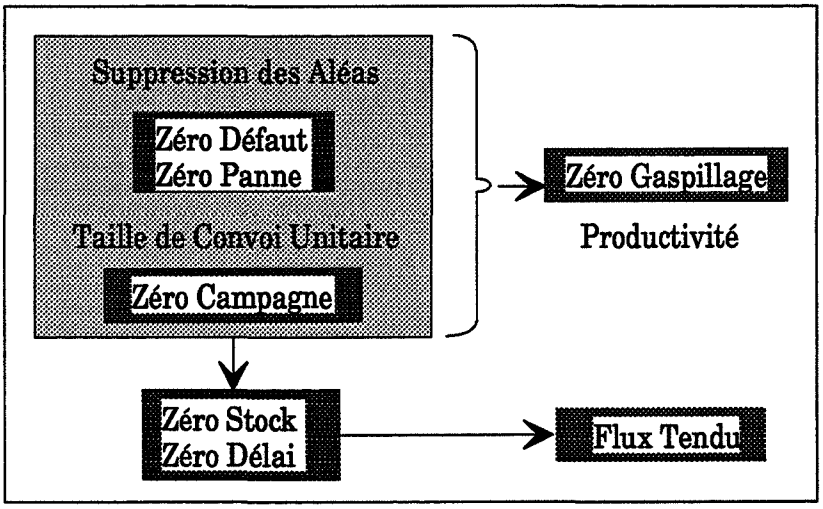


Figure 1-4 Des Cinq Zéros au Flux Tendu

En effet, le juste-à-temps ne remet pas en cause l'organisation traditionnelle des systèmes de programmation qui, sur la base d'hypothèses moyennes concernant les temps de cycle de production, les taux de pannes des moyens de production, les rebuts et retouches, calculent des programmes à exécuter chaque jour et dans chaque équipe... En conséquence, des suivis de production sont nécessaires pour comparer les réalisations par rapport à la théorie. Là

commencent les différences. Il va falloir gérer une organisation d'atelier où les hommes vont courir après les écarts, le plus souvent sans utilité réelle. La principale différence entre la méthode M.R.P. et la méthode juste-à-temps est résumée sur le tableau 1-1.

Dans le cas de M.R.P., on *court* après les écarts, on les constate et on les actualise a posteriori lorsque le programme est respecté, l'écart est payé dans les coûts d'exploitation de l'atelier. On subit une augmentation des coûts non prévus, c'est la fatalité des écarts, et on réfléchit.

Dans le deuxième cas, on va sécuriser et surveiller a priori le fonctionnement par rapport au référentiel: taux de pannes, rebuts-retouches, temps cycles techniques, etc. Il en résultera une réduction des coûts non prévus et un refus de la fatalité de dysfonctionnements durables.

LOGIQUE M.R.P.	LOGIQUE JUSTE-A-TEMPS
- calculs d'ordres "moyens" - course permanente entre "réalisé" et "prévu" - les variations de programme sont amplifiées par des ajustements de stocks - les ajustements de stocks génèrent des fluctuations perturbatrices sur plusieurs périodes ⇓ **CALCULS COUTEUX et GASPILLAGES**	- génération des ordres par flux physiques et commandes - pas de course entre "réalisé" et "prévu": *visibilité permanente à l'atelier* - lissage des variations quantitatives - lissage des fluctuations perturbatrices ultérieures ⇓ **CALCULS REDUITS et SUPPRESSION DES GASPILLAGES**
On constate **A POSTERIORI** les écarts et on actualise les données ⇓ **DYSFONCTIONNEMENT et ACTION CURATIVE**	On surveille **A PRIORI** le fonctionnement par rapport au référentiel ⇓ **SURVEILLANCE et ACTION PREVENTIVE**
Réduction des coûts "non prévus" signifie la fatalité des écarts que l'on se justifie et l'on réfléchit	Réduction des coûts "non prévus" signifie le refus de la fatalité que l'on sécurise et l'on surveille

Tableau 1-1 Comparaison des Méthodes M.R.P. et Juste-A-Temps

III.2.2 La Méthode Kanban

La méthode Kanban (Kanban veut dire étiquette en japonais) est généralement confondue avec la logique juste-à-temps. Le juste-à-temps concerne tous les échelons de l'entreprise, c'est une chasse au gaspillage [BERANGER 87] alors que le Kanban est une méthode de gestion des flux uniquement issue de la logique juste-à-temps.

Un Kanban est un ordre de fabrication implicite correspondant à une opération:

- définie du processus de fabrication, pour une référence précise du produit;
- effectuée sur un poste de travail déterminé;
- pour une quantité de pièces fixe, contenue généralement dans un conteneur.

Outre les Kanbans de fabrication qui circulent entre les centres de fabrication et l'aire de stockage situé en aval de ce centre, les Kanbans de transfert sont des étiquettes qui circulent entre l'aire de stockage et les centres demandeurs (figure 1-5).

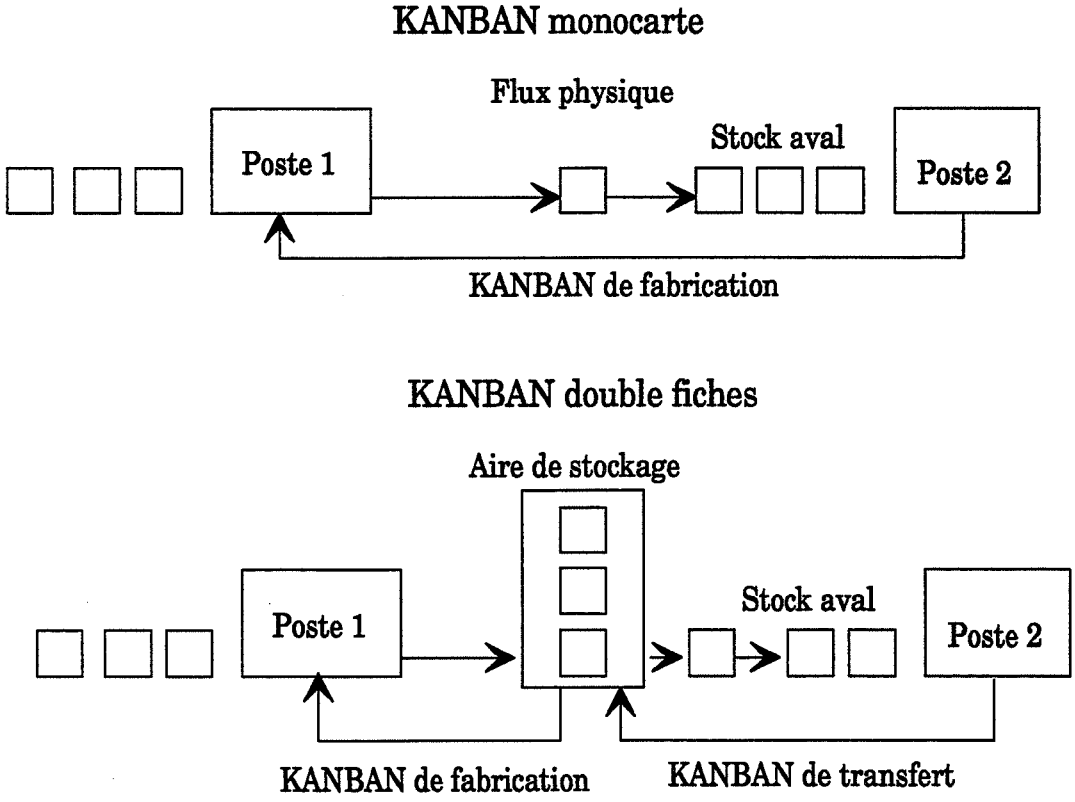


Figure 1-5 Méthode Kanban

La méthode Kanban a pour effet d'assurer une circulation de l'information de l'aval vers l'amont. Les mises en oeuvre de fabrication en amont se trouvent pilotées par les fabrications

réelles de l'aval. Le flux de production se trouve plus ou moins tendu suivant le nombre de Kanbans présents dans le système. En effet, puisque aucun conteneur ne peut être produit en l'absence de Kanban, le nombre de conteneurs d'une référence en attente à un poste ne peut dépasser le nombre de Kanbans. Le volume du flux est donc régularisé par le nombre de Kanbans. En diminuant le nombre de Kanbans, on diminue automatiquement le niveau des en-cours de la référence.

En résumé, la méthode Kanban vise en priorité à augmenter la réactivité (flux tendu) du système de production de la manière suivante:

- la circulation simplifiée du flux matière est obtenue par une réimplantation des moyens de production (i.e. îlot de fabrication, cellule flexible). Cette implantation suppose une bonne standardisation des produits, et plus généralement une application de la technologie de groupe.

- le faible débit du flux matière est obtenu par la réduction de la taille des lots de fabrication, et la livraison des matières premières et composants en quantités variables et juste-à-temps. Ceci suppose une forte adaptabilité des éléments qui concourent à la production, sous-traitants, fournisseurs inclus.

- l'homogénéité de la nature du flux matière est obtenue par la standardisation des produits et leurs fabrications en grande série. La méthode Kanban encourage la réduction de la taille des lots en supposant une forte adaptabilité des moyens de production: réglages rapides, changements rapides d'outillages, polyvalence, etc.

- le suivi et le pilotage de la production sont décentralisés et simplifiés. Les flux d'information qui assurent la régulation du flux matière sont constitués des Kanbans.

III.2.3 Conclusion sur la Méthode Juste-A-Temps et la Méthode Kanban

Les résultats obtenus par la méthode juste-à-temps et par la méthode Kanban sont toujours particulièrement spectaculaires et rapides [MILLE 93]:

- Suppression des systèmes de calculs de programmes à court terme et de toute l'organisation qui distribue les programmes et court après les écarts quotidiens et leurs justifications.

- Suppression des perturbations par une meilleure utilisation des moyens industriels.

- Réduction des stocks et en-cours par la gestion des priorités de réalisation ou règles de pilotages (et non plus des quantités-dates).

- Amélioration de la visibilité dans l'atelier des ordres à réaliser, ce qui va accroître la réactivité de l'organisation face aux problèmes rencontrés (Les ordres n'iront plus se perdre sur des états informatiques dans des bureaux).

- Réduction drastique et rapide des anomalies de fonctionnement de l'atelier et une amélioration permanente de la qualité du fonctionnement du système (qualité totale).

Cependant, les conditions d'utilisation de cette méthode font qu'elle ne peut être appliquée à tous les systèmes de production. Une bonne circulation du flux nécessite une organisation du système par lignes de produits et une demande suffisamment régulière. Puisque la méthode déclenche la fabrication sur la base de consommations passées (historiques). Cela implique une consommation régulière et connue. Sinon, la constitution ou l'accroissement de stocks de sécurité serait nécessaire, d'où une perte de l'intérêt de la méthode [WALDNER 90].

III.3 La Méthode O.P.T.

La méthode O.P.T. (Optimized Production Technology) est due à la société CREATIVE OUTPUT fondée principalement par Goldratt. Elle est née d'une réflexion critique sur de nouveaux objectifs pour la gestion de production [BENASSY 90]:

- augmenter le produit des ventes, c'est-à-dire l'argent généré par les ventes;
- diminuer les dépenses d'exploitation, c'est-à-dire l'argent dépensé pour produire;
- et augmenter la trésorerie, c'est-à-dire retarder l'engagement d'argent pour produire (retarder le temps d'achat de matières premières), accélérer le retour sur engagement.

A l'heure actuelle, l'évaluation économique en gestion de production est en pleine mutation. La méthode O.P.T. est une illustration de cette mutation. En effet, si les méthodes de gestion de production ont évolué vers des logiques plus aptes à répondre au nouvel environnement économique, elles n'ont peu ou pas changé d'indicateurs économiques. La plupart des outils comptables actuellement disponibles ont été mis au point avant 1925 à des fins de planification et de contrôle dans de grandes entreprises américaines [BRODIER 88]. Le calcul des coûts des produits, des coûts de production, la comptabilité analytique en général, n'ont ainsi pas évolué depuis l'époque du taylorisme où les méthodes de travail étaient totalement différentes.

Goldratt, à l'origine de la méthode O.P.T., et Kaplan spécialiste reconnu de l'évaluation économique en gestion de production, affirment respectivement que *la comptabilité est l'ennemi numéro 1 de la productivité, la comptabilité d'hier sape la production* [GIARD 88].

La méthode O.P.T. est pour cela une méthode de gestion de production proposant de nouveaux indicateurs économiques. Elle considère en premier lieu que l'élaboration d'un plan de production consiste à satisfaire *simultanément* des contraintes de nature différente. Ces contraintes sont d'ordre technique (la capacité d'un moyen de production par exemple), d'ordre économique (la valeur ajoutée à un produit par exemple) et d'ordre externe (la commande d'un client par exemple). Cependant, deux idées étoffent cette logique:

- Toutes ces contraintes ne sont pas indépendantes parce que les événements de la production ne sont eux-mêmes pas indépendants. Ainsi par exemple, la contrainte de capacité d'un moyen de production ne pose pas globalement un problème dans l'élaboration du plan de production, si les événements de la production montrent que le moyen de production n'est jamais saturé. Satisfaire cette contrainte localement (saturer le moyen de production en vue de l'employer pleinement) revient à augmenter le débit matière du moyen de production amont qui l'alimente.

- Une minorité de ces contraintes conditionne l'obtention d'un plan de production optimisé. L'expérience montre que ce sont celles induites par les moyens de production *goulots d'étranglement*. On peut définir un moyen de production goulot comme un moyen dont le rapport charge/capacité sur une période est souvent supérieur à 1.

C'est sur la base de ces deux idées fondamentales que l'application de la méthode O.P.T. trouve un sens. Les principes de cette méthode relativement récente (énoncés en 1979) sont résumés dans le tableau1-2. Ces règles ont été très bien décrites dans [GOLDRATT 86].

Les règles 1 et 2 sont parmi les règles qui allaient à l'encontre de l'idée qui était généralement admise et qui consistait à considérer que la maximisation des taux d'utilisation de toutes les ressources est un objectif en soi. La méthode O.P.T différencie nettement les ressources goulots des ressources non-goulots et suggère de maximiser le taux d'utilisation des ressources goulots seulement. Optimiser les ressources non-goulots conduirait à une surproduction et donc à des stocks inutiles. La méthode O.P.T. favorise l'équilibrage du flux matière qui consiste à fixer un débit du flux matière en respectant les contraintes de capacité de chaque moyen de production. Car équilibrer la capacité de production à court terme est une tâche difficile, voire impossible:

N°	Règles O.P.T.	Règles classiques
1	Equilibrer les flux, non les capacités	Equilibrer les capacités puis essayer d'équilibrer les flux
2	Le niveau d'utilisation d'une ressource non goulot n'est pas déterminé par son propre potentiel mais par d'autres contraintes du système	Le niveau d'utilisation d'une ressource est déterminé par sa propre capacité
3	Utilisation et activation d'une ressource sont à distinguer	Utilisation et activation d'une ressource sont synonymes
4	Une heure perdue sur un goulot est une heure perdue sur l'ensemble du système	Une heure perdue sur un goulot est seulement une heure perdue sur cette ressource
5	Une heure gagnée sur une ressource non goulot ne rapporte rien	Une heure gagnée sur une ressource est une heure gagnée quelle que soit la ressource
6	Les goulots déterminent à la fois le débit de sortie et le niveau des stocks	Les goulots limitent temporairement le débit de sortie mais ont peu d'effet sur les niveaux des stocks
7	Le lot de transfert peut, et souvent doit, ne pas être égal au lot de fabrication	Il faut éviter l'éclatement et le chevauchement des lots lancés
8	Les lots de fabrication peuvent varier selon les opérations	Un lot lancé doit rester entier
9	Les ordonnancements doivent être faits en considérant l'ensemble des contraintes du système. Les temps d'attente sont des variables dynamiques et non des données	Les ordonnancements doivent être effectués par la suite d'actions suivantes: 1) détermination de la taille des lots, 2) calcul des délais, 3) désignation des priorités en fonction des délais, 4) ajustement des ordonnancements successifs aux contraintes apparentes de capacité
10	La somme des optimums locaux n'est pas l'optimum global	La seule manière d'atteindre un optimum global est de rechercher les optimums locaux

Tableau 1-2 Comparaison des Règles de la Méthode O.P.T et Classiques.

- La capacité est sujette aux aléas de fabrication, donc est en permanence déstabilisée;

- la capacité varie généralement par valeurs discrètes qui ne répondent pas nécessairement avec exactitude aux besoins réels de l'équilibrage;

- la demande varie selon un ordre de grandeur temporel incompatible avec l'intervalle de variation de la capacité.

Les règles 4, 5 et 6 signifient que la productivité du système est déterminée par les ressources goulots et non pas par toutes les ressources prises indépendamment.

Les règles 7 et 8 remettent en cause le principe de quantité économique qui fixe la même taille pour les lots à fabriquer et à transférer. Le transfert de quantités plus faibles que celles des lots de fabrication permet d'accélérer la mise à disposition des produits devant les moyens de production goulots, alors que la taille des lots de fabrication est plus grande pour les moyens de production goulots afin d'en diminuer les réglages, les changements d'outillages, etc. Cela n'est pas nécessaire pour un moyen de production non-goulot, car par nature, son niveau d'utilisation lui assure une certaine disponibilité. Son plein emploi n'est donc pas un objectif majeur, et son utilisation peut s'accommoder d'une taille de lots de fabrication plus petite. Il faut donc distinguer l'utilisation des ressources non-goulots et le plein emploi des ressources goulots (règle 3).

La règle 9 préconise de prendre en compte les contraintes matières (ordres de fabrication à réaliser) et les contraintes de capacités de moyens de production simultanément. Les délais de fabrication sont le résultat d'un programme et ne peuvent donc pas être prédéterminés.

La devise de la méthode O.P.T. résumant ainsi l'ensemble des 9 règles énoncées, est que la somme des optimums locaux n'est pas l'optimum global du système (règle 10).

Conclusion sur la Méthode O.P.T.

La méthode O.P.T. est un système de gestion par contraintes qui met l'accent sur la prise en compte simultanée de contraintes de natures différentes. Les règles de la méthode O.P.T. paraissent à première vue évidentes, pourtant c'est bien souvent le contraire qui est mis en pratique et même enseigné.

La méthode O.P.T. dépasse le cadre d'une méthode de gestion de production classique puisqu'elle fournit une technique de représentation du système de production. Cette technique, malheureusement mal connue, consiste à décrire un réseau en y intégrant des contraintes de

toute nature. L'objectif de cette représentation est de dissocier les moyens de production goulots des non-goulots en vue de l'application des règles de la méthode.

Le secret qui entoure l'algorithme d'ordonnancement fourni avec la méthode ne permet pas une comparaison directe avec les autres méthodes. Quant à son utilisation en France, elle reste très limitée. En 1989, on ne dénombrait, d'après [BENASSY 90], qu'un seul utilisateur qui en est d'ailleurs satisfait (même il n'y a plus maintenant).

III.4 Conclusion sur les Méthodes de Gestion de Production

La méthode M.R.P. privilégie la gestion des articles dans le temps: les produits finis résultent de l'assemblage de composants, fabriqués par l'entreprise ou par des sous-traitants. C'est la "nomenclature de production". L'atelier est décomposé en postes de charge, c'est-à-dire, une ou plusieurs machines qui réalisent le produit selon sa gamme de fabrication définissant la suite des opérations. Cette méthode propose donc de partir de prévisions de production sur les produits finaux pour calculer les besoins en composants.

La méthode O.P.T. remet en cause les approches classiques de gestion de la production. Fondée sur une planification au plus serré des moyens de production goulots d'étranglement (anti-flux) et sur une simulation économique des choix d'ordonnancement résultants, cette méthode équilibre les flux de matières et non les capacités de moyens de la production. Son logiciel, très cher, reste pour la plupart des spécialistes assez mystérieux et peu utilisé dans l'industrie.

Le concept juste-à-temps cherche, grâce à une organisation adaptée de la production, à fabriquer et à livrer les produits au bon moment. Le système utilise des fiches Kanban, plaquées sur les flux physiques de production. Très répandue au Japon, cette méthode trouve ses limites dans le cas d'une production en petite série.

Pour le plus grand bien des utilisateurs potentiels à la recherche du moyen idéal pour gérer leur production, ces trois approches, souvent mises en concurrence, apparaissent aujourd'hui complémentaires et convergentes. La méthode M.R.P., par exemple, est basée sur une production hebdomadaire; elle n'est donc pas adaptée au contrôle de production (à moins qu'elle soit très stable, ce qui est rare) ni à l'amélioration des flux de produits sur les lignes de fabrication. Par contre, le juste-à-temps s'applique à une production quotidienne, voire horaire par la parfaite synchronisation des différentes opérations concernées. Or comme Goldratt l'a constaté [SCHERER 90]: "synchroniser les différentes opérations comme le fait le J.A.T., ce n'est pas suffisant: il faut auparavant détecter et éliminer les goulots dans le flux de

production." Selon lui, les chaînes de production comportent toujours des maillons faibles. Il suffit donc de détecter les faiblesses du système de production pour l'améliorer. Alors, M.R.P., J.A.T. ou O.P.T., quel système choisir? Pas facile de décider, on trouve souvent dans les trois méthodes le tout et son contraire. Il ne faut donc pas à notre sens chercher à comparer les logiques et les méthodes, mais chercher une manière de les intégrer. Pour bien réussir une GPAO, il suffit de prendre le meilleur de chaque concept et d'écarter l'inutile? Facile-à-dire...

IV Conclusion

Ce chapitre a eu pour objectif de présenter en général les systèmes de production et leur gestion. Différentes décisions à prendre au sein de la gestion sont décrites et hiérarchisées. Les méthodes de gestion de production les plus utilisées sont présentées et analysées.

La présentation de ces différents points a pour but de fixer le cadre général de notre travail et de décrire les différentes approches ou méthodes de modélisation que nous serons amenés à utiliser pour construire des modèles de simulation. Avant d'aborder les problèmes de modélisation et simulation des systèmes de production dans le chapitre II (un moyen pour comprendre et maîtriser la complexité des systèmes de production et de sa gestion), on peut dire qu'il y a deux fortes tendances dans le domaine de la gestion de production: l'intégration des systèmes de production et l'application des concepts d'ingénierie simultanée dans la gestion.

Actuellement, plusieurs auteurs mettent l'accent sur le concept d'**intégration** ou d'**entreprise communicante et intégrée** ([COMMI 90], [WALDNER 90]). L'intégration aboutit au concept d'entreprise globale dans laquelle toutes les fonctions sont couplées et interconnectées sur un même système de communication; elle réagit en temps réel (sans délai, sans inertie); elle gère de l'information, et les processus de décision tendent à l'optimiser comme un système global et non comme une juxtaposition de fonctions autonomes.

Le concept d'entreprise intégrée a pour but de procurer à l'ensemble des acteurs de l'entreprise une vision pertinente du système dans lequel ils évoluent: une vision synthétique et globale pour les dirigeants; une vision spécialisée et détaillée pour les personnels opérationnels. Depuis quelques années, une démarche d'**ingénierie simultanée** est préconisée. Cette démarche consiste à intégrer les activités de conception et d'industrialisation des produits en prise en compte de la maintenance sur la totalité du cycle de vie afin d'améliorer la qualité et la réactivité dans l'entreprise et de diminuer les coûts et les délais de production, etc. [MORIN et al. 1993].

L'**ingénierie simultanée** est une approche systémique qui intègre le développement simultané des produits et des processus associés, incluant la fabrication et le soutien logistique. Cette approche prend en considération le démarrage, le cycle de vie du produit depuis sa conception jusqu'à son exploitation, incluant la qualité, les coûts, la planification et les besoins utilisateurs. Cette démarche, à notre sens, va certainement remettre en cause ou modifier les méthodes de gestion traditionnelles: organisation des systèmes de production (séquentiel ---> parallélisation), communication entre les partenaires dans l'entrepise (travail en groupe pluridisciplinaire), tendance du marché ou exigences clients (satisfaction du besoin, qualité globale), développement et innovation du produit (centralisation des données techniques), etc. [BOURDICHON 93].

Chapitre II Modélisation et Simulation des Systèmes de Production

I Introduction

Le problème de la modélisation des systèmes de production consiste à trouver une correspondance entre le monde réel et un modèle informatique. Dans le domaine de la productique, ce problème est d'autant plus difficile à appréhender que les systèmes de production et de gestion à représenter ne sont pas formels. La complexité et la diversité de ces systèmes rendent d'autant plus difficile la détermination d'un modèle idéal.

Il y a trois composants dans le processus de modélisation des systèmes de production: le système de production, le modélisateur et le modèle. Le modèle est une représentation du système conçue par le modélisateur pour certains objectifs. Le système à représenter est donc la référence du modèle. Un modélisateur construit son modèle en s'appuyant sur sa perception partielle de la référence et à l'aide d'une boîte à outils (un ensemble de types de modèle appelé couramment plus brièvement modèle) (figure 2-1). Une définition descriptive du modèle est donc [EVAN 88]:

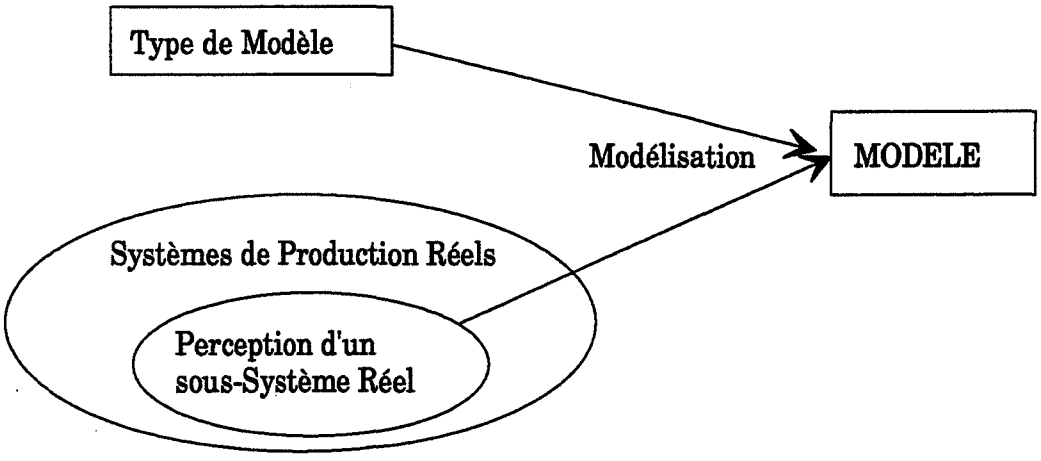


Figure 2-1 Modélisation (Représentation) du Problème

Un modèle est une structuration simplifiée qui représente censément des caractéristiques et des relations signifiantes de la réalité sous une forme généralisée. Les modèles sont des approximations subjectives de haut niveau qui n'incluent pas toutes les observations ou

mesures associées, mais font apparaître les aspects fondamentaux de la réalité tout en ignorant certains détails sécondaires.

Le mot "modèle" comporte deux aspects essentiels: un modèle peut refléter la réalité et prédire les événements et les phénomènes réels qui se heurtent à nos idées, ou il peut servir comme une forme (référence) idéale ou un phénix (parangon) avec lequel nous voulons agir sur la réalité (figure 2-2) [EVAN 88].

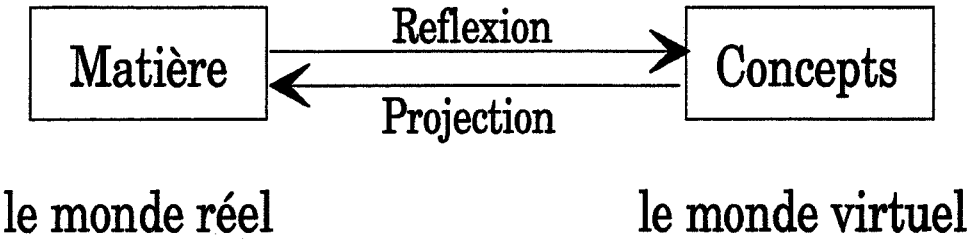


Figure 2-2 Vues Idéales et Matérialistes des Modèles

L'application des techniques de modélisation peut donc être effectuée de deux façons. Nous pouvons décrire ou "refléter" un aspect particulier du monde réel; dans ce cas, la modélisation joue un rôle explicatif fondamental dans la compréhension des systèmes de production. Alternativement, nous sommes en présumé d'un genre de modèle quand les modélisateurs essaient de démontrer comment le monde réel doit être. La vision artistique et la conviction politique tombent dans cette catégorie normative de modèles. Dans toute modélisation, un aspect en domine habituellement un autre mais jamais exclusivement. Nous nous préoccupons dans ce travail essentiellement du premier aspect: essayer de modéliser les systèmes de production avec des modèles et des méthodes adéquats.

De nombreux types de modèles sont utilisables dans la modélisation des systèmes de production. Ils peuvent être classés de plusieurs façons: ils peuvent être statiques ou dynamiques; ils peuvent être décrits en fonction des matériaux qui les constituent ou en fonction de la nature abstraite qu'ils transforment: théoriques ou symboliques, mentaux ou conceptuels. Les modèles physiques peuvent être iconiques ou analogiques; les modèles mathématiques peuvent être déterministes ou stochastiques. La simulation des systèmes de production est un processus ou une technique de modélisation dans lequel le dynamisme de la réalité, soit actuel soit projeté du système, est imité par un modèle stochastique et déterministe. Différents types de modèle existent autant dans le domaine informatique que dans le domaine productique, par exemple, réseau de PETRI [BRAMS 83], modèle mathématique et GRAFCET [BLANCHARD 79] dans l'automatique, modèles SA, SD SADT, SART, REMORA et MERISE [JAULENT 92] dans l'informatique et la productique, modèles IDEF,

GRAI [PIERREVAL 90], OLYMPIOS ([BRAESCH 89], [MAIRE 91]), AMS [MELESE 79] et modèle CIMOSA [AMICE 89] dans la gestion de production, etc.

Pour mettre en oeuvre un modèle, on a besoin d'une méthode ou d'une démarche d'analyse et de conception de modèle. Deux grandes classes de méthodes existent [ROLLAND 86]:

- les méthodes et modèles s'appuyant sur une approche fonctionnelle, qui consistent à considérer le système d'information par les traitements qu'il doit exécuter plutôt que par les données qu'il doit gérer (SADT, IDEF0, GRAI, PETRI, etc.).

- les méthodes s'appuyant sur une démarche systémique, qui consistent à considérer le système d'information comme un modèle de la réalité organisationnelle (MERISE, REMORA, OLYMPIOS, AMS, etc.).

II La Méthode SADT

SADT [IGL 89] (Structured Analysis and Design Technique est une méthode ou un langage pluridisciplinaire de spécification fonctionnelle de systèmes. Elle n'est pas à proprement parler une méthode de conception puisqu'elle se limite à une spécification d'un système en ignorant les aspects liés à la réalisation de ce système. Cette méthode est souvent employée au préalable à la réalisation du schéma directeur, pour mieux connaître la situation existante et dégager les fonctions souhaitées du système à concevoir. C'est une méthode générale. La variété de ses domaines d'application en témoigne: génie logiciel, productique, système de gestion, aéronautique, etc.

La méthode SADT fournit plusieurs représentations graphiques formalisées pour analyser et comprendre un système à construire:

- le modèle SADT du système existant,
- le modèle SADT du système idéal,
- le modèle SADT du système tel qu'il est réalisable,
- le modèle SADT du système futur (tel qu'il sera réalisé).

Lors de la réalisation de ces modèles, l'accent est porté sur la spécification des fonctions que le système remplit actuellement (système existant), devrait remplir (système idéal), serait effectivement capable de remplir (système tel qu'il est réalisable) et remplira (système futur). Ces fonctions sont décrites indépendamment de la manière dont elles sont, devraient être, pourraient être ou seront réalisées [LISSANDRE 90].

II.1 Les Concepts de la Méthode

Conçue à partir de concepts simples et basée sur un formalisme graphique et textuel facile à apprendre, la méthode SADT permet d'une part de modéliser le problème avant de chercher à en exposer une solution et, d'autre part, d'assurer une communication efficace entre les différentes personnes concernées par le système à construire. Sept concepts fondamentaux sont à la base de la méthode SADT [JAULENT 92]:

1. SADT analyse un système en construisant un modèle de celui-ci, dans le but d'en exprimer une compréhension complète et de la situer dans son *contexte*, c'est-à-dire d'en préciser le *sujet*. Plusieurs modèles exprimant différents *points de vue* (utilisateur, constructeur, gestionnaire, responsable de maintenance...) sur le système peuvent se révéler nécessaires.

2. SADT analyse un système de manière *descendante*, *modulaire*, *hiérarchique* et *structurée*.

3. SADT distingue la *description fonctionnelle* du système (quoi faire?), domaine où elle est particulièrement efficace, de la *description des différentes solutions* envisagées pour sa réalisation (comment faire?).

4. SADT privilégie deux aspects dans la modélisation du système à décrire. Le premier aspect est relatif aux transformations, fonctions, activités, tâches, traitements, etc., qui s'effectuent dans le système. Il est pris en compte par SADT dans la notion d'***activité***. Le second aspect concerne les objets manipulés par les activités. On parle plus généralement de ***données*** dans la méthode SADT.

5. SADT est basé sur des principes de représentation graphique (un langage pluridisciplinaire *semi-formel*) destinés à faciliter la compréhension du système spécifié. Pour cela, les modèles SADT sont représentés graphiquement par des boîtes inter-reliées.

6. SADT favorise un *travail d'équipe* (auteurs, lecteurs, experts) discipliné et coordonné. Elle permet de mettre en évidence les résultats qui reflètent le mieux la qualité du travail.

7. SADT oblige à *consigner sous forme écrite* tous les choix et décisions faits pendant l'analyse. Les documents créés sont accessibles à tous pour être revus et corrigés.

II.2 Les Outils de Modélisation

SADT adopte deux types de représentations duales (figure 2-3). Dans la première, les activités sont représentées graphiquement dans un modèle par des *boîtes*, désignées par des *verbes*, éventuellement munies de compléments. Les données seront, quant à elles, représentées par des *flèches*, désignées par des *noms*. Les diagrammes obtenus sont appelés des *actigrammes*. Dans la seconde représentation, les boîtes représentent des données et les flèches des activités. Les diagrammes obtenus sont des *datagrammes*. Bien qu'ils permettent d'effectuer des "références croisées" avec les actigrammes, les datagrammes semblent moins utilisés. C'est pourquoi, par la suite, nous ne nous intéresserons qu'aux actigrammes, seuls retenus par la méthode IDEF. Les datagrammes sont représentés par des modèles MERISE que nous proposerons dans le prochain paragraphe.

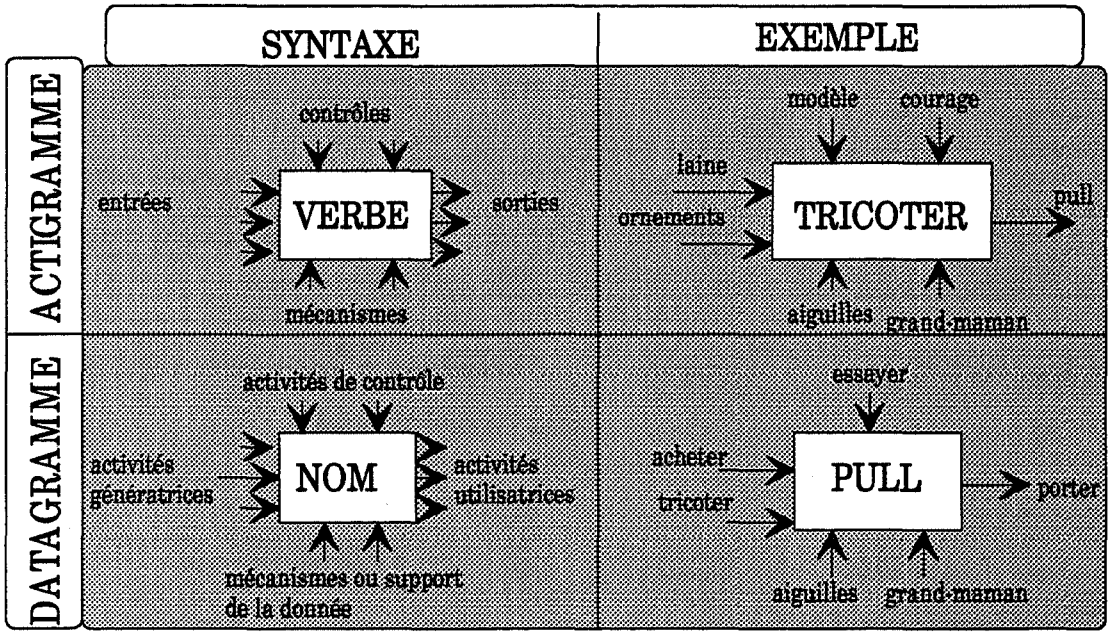


Figure 2-3 Actigramme et Datagramme du Modèle SADT

Les *entrées* sont des données consommées ou converties par l'activité pour produire des *sorties*. Les données de *contrôle* expriment les conditions et/ou les circonstances qui gouvernent, orientent, ou contraignent l'activité. Ce sont, en général, soit des données qui déclenchent ou inhibent la transformation, soit des paramètres ou des valeurs de consigne qui la régissent. Les données *mécanismes* précisent la personne, le service, la ressource ou tout autre élément qui supporte ou effectue la fonction. Ces données expriment en général le *comment* ou le *qui*.

Un modèle SADT est composé d'un ensemble de diagrammes ordonnés hiérarchiquement. Au niveau le plus général, nous trouvons le diagramme de *contexte, de niveau -1* (schéma A0 dans

la figure 2-4), qui montre les sources et les destinations des différentes informations arrivant ou sortant de la "boîte à analyser". La boîte à analyser correspond quant à elle, au diagramme de niveau inférieur noté : *niveau -0* (schéma A1 dans la figure 3-26), elle même décomposable en niveau 0, 1, 2... Chacun de ces diagrammes peut être considéré, soit comme un diagramme-père, synthèse de ses diagrammes-enfants, soit comme un diagramme-enfant, analyse d'une partie de son diagramme-père.

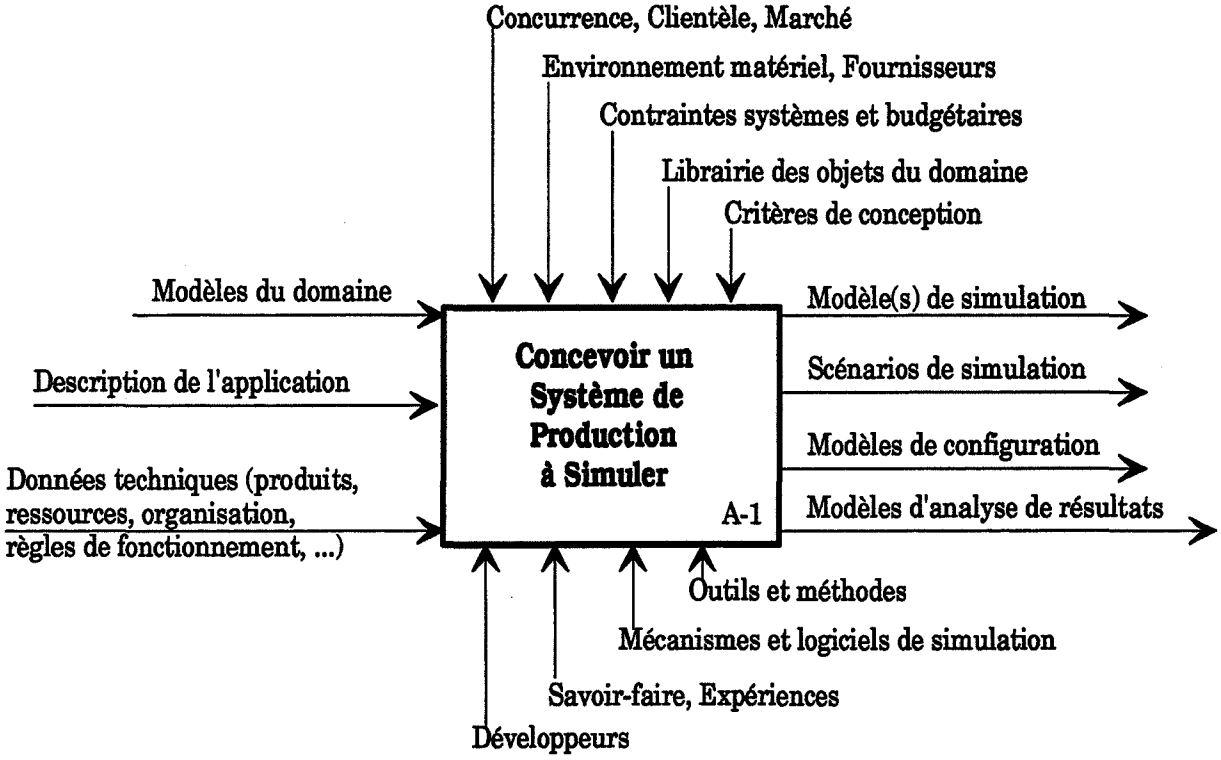


Figure 2-4 Schéma SADT Contexte de Développement

II.3 La Démarche de Modélisation

Recherchant avant tout la rigueur, la méthode SADT fournit de façon très détaillée, par l'intermédiaire de ses concepts, l'ensemble des démarches nécessaires à la mise en pratique d'une analyse et d'une conception. La démarche de modélisation permet à un ou plusieurs auteurs de construire le modèle d'un système suivant 7 étapes.

Etape 1: Préparation du modèle. Cette étape vise à définir le sujet, le but et le point de vue du modèle à construire.

Etape 2: Construction de la liste de données. Dans cette étape, l'auteur collecte le maximum de données sur le sujet du modèle. Ces données correspondent aux besoins, aux caractéristiques et aux contraintes du sujet du modèle.

Etape 3: Construction de la liste d'activités. Pour chaque donnée de la liste précédemment établie, l'auteur identifie les activités créant, utilisant ou modifiant cette donnée. L'ensemble des activités obtenu constitue la liste des activités.

Etape 4: Construction du diagramme d'activités.

Etape 5: Vérification de la qualité des diagrammes d'activités. La qualité se juge par son facteur d'amplification, son respect du point de vue, son but, l'équilibre de ses niveaux de détail et sa complétude.

Etape 6: Elaboration de documents complémentaires aux diagrammes d'activités.

Etape 7: Couplage des diagrammes d'activités à un modèle entité-association.

II.4 Conclusion sur la Méthode SADT

Malgré quelques lacunes (absence de spécification du point de vue des performances et des contraintes temporelles, cohérence entre actigrammes et datagrammes douteuse, point de vue dynamique très limité...), la méthode dispose de nombreuses qualités telles que:

- sa très bonne lisibilité grâce à la clarté de ses graphiques,
- le nombre restreint de ces concepts de base, assurant un apprentissage facile,
- son aptitude à communiquer dans un langage non "informatique",
- l'organisation de l'équipe qui réalise la modélisation,
- sa structure hiérarchique et modulaire du modèle obtenu.

La méthode SADT, créée à l'origine "pour communiquer des idées" doit être utilisée pour exprimer des besoins (pas nécessairement informatiques), aborder l'aspect fonctionnel d'un cahier des charges, communiquer entre les différents membres d'une équipe, mais ne peut pas servir en tant qu'une méthode d'analyse fonctionnelle de logiciel (et encore moins pour de logiciel temps réel), car les résultats obtenus dépendent plus de la compétence de l'analyste que de la rigueur de la méthode. Pour cela, des extensions ont été définies, d'une part pour lever les ambiguïtés qui subsistent au niveau des données: un passage au modèle entité-association dont nous parlerons dans la méthode MERISE, d'autre part pour modéliser la dynamique d'un logiciel à fortes contraintes temps réel: un passage au modèle SART [HATLEY et al. 91] qui permet l'élaboration d'un modèle en pensant "réponse à des événements", provenant de l'"espace action-perception" du système. Malgré que la méthode SART soit adaptée à la spécification fonctionnelle et dynamique de système, elle est difficile de construire un modèle

correspondant. Nous ne présenterons donc pas la méthode SART, mais la méthode SADT est bien utilisée dans la phase "analyse de l'application" (chapitre III).

III La Méthode MERISE

La méthode MERISE s'appuie sur de nombreux principes issus de la systémique. Son objectif est de fournir à la fois une philosophie, une démarche, des modèles, des formalismes et des normes, pour concevoir et réaliser un système d'information. Cette méthode couvre l'ensemble des phases du cycle de vie du système d'information: de l'analyse des besoins jusqu'à la maintenance.

III.1 Les Concepts de la Méthode

Une vision globale

La méthode MERISE met en évidence la nécessité d'une conception globale du système d'information pour éviter les problèmes liés à la redondance des données dans l'entreprise, l'absence de cohérence dans leur mise à jour et la difficulté de maintenir de façon convenable le système d'information. Le système d'information conçu par cette méthode s'inscrit parfaitement dans le schéma de l'approche systémique de l'entreprise [MELESE 79] proposée par Le Moigne [MOIGNE 90]. Ce schéma illustre une décomposition du système-entreprise en trois sous-systèmes: le système opérant, le système de décision et le système d'information.

La séparation des données et des traitements

La méthode MERISE propose deux représentations du système d'information. L'une, fournie par les *données*, donne une vision statique du système-entreprise. L'autre, fournie par les *traitements* sur ces données, donne une vision dynamique de ce système. Cette distinction entre les aspects statique et dynamique doit permettre entre autres une meilleure adaptation du système d'information lors de l'évolution des besoins [PIERREVAL 90].

Une approche par niveaux

La méthode MERISE met en évidence trois niveaux dans la conception du système d'information: niveau conceptuel, niveau organisationnel et niveau technique. Ces trois niveaux sont successivement abordés aussi bien dans la description des données que dans celle des traitements.

III.2 Les Outils de Modélisation

La méthode MERISE fournit six modèles (tableau 2-1) permettant de représenter les données et les traitements sur les trois niveaux d'abstractions.

NIVEAUX	MODELES	
	DONNEES	TRAITEMENTS
Niveau Conceptuel	Conceptuel MCD	Conceptuel MCT
Niveau Organisationnel	Logique MLD	Organisationnel MLT
Niveau Physique	Physique MPD	Opérationnel MPT

Tableau 2-1 Modèles de la Méthode MERISE

Le niveau *conceptuel* permet la description des finalités de l'entreprise (objectifs généraux et contraintes). Il apporte la réponse à la question "quoi?" (aspect fonctionnel) et représente le niveau le plus stable du système-entreprise.

Le niveau *organisationnel* répond aux questions "qui, où et quand?". Il décrit la structure à mettre en place pour satisfaire les objectifs décrits au niveau précédent, et représente un deuxième niveau de variabilité du système.

Le niveau *technique* définit les moyens techniques à mettre en oeuvre pour réaliser le système d'information. Il répond à la question "comment?". Parce qu'il subit les évolutions fréquentes des matériels et logiciels, il correspond au niveau le moins stable du système-entreprise.

La méthode MERISE possède différents outils pour la construction des modèles. Le formalisme le plus utilisé est le type *entité-association*. Décrit avec ce formalisme, les modèles se composent d'*entités* (ou classes d'entités) qui caractérisent un certain type d'objets (que l'on appelle aussi: individus, éléments ou constituants de la base de données). Les entités sont repérées par leurs *identifiants*. Elles possèdent un certain nombre de *propriétés*. Une propriété est une donnée élémentaire (un attribut en terme d'objet) décrivant l'entité.

Les entités sont reliées par des relations. Les relations peuvent être binaires ou n-aires selon qu'elles relient deux ou plusieurs entités. Il existe des relations internes à une même classe d'entité. Des cardinalités sont associées aux relations. On les définit comme le nombre d'occurrences de la relation pouvant exister pour une occurrence de l'objet (nombre minimum et maximum d'occurrences). Figure 2-6 illustre un modèle entité-association étendu de traitement de commande selon la logique gestion par prévision.

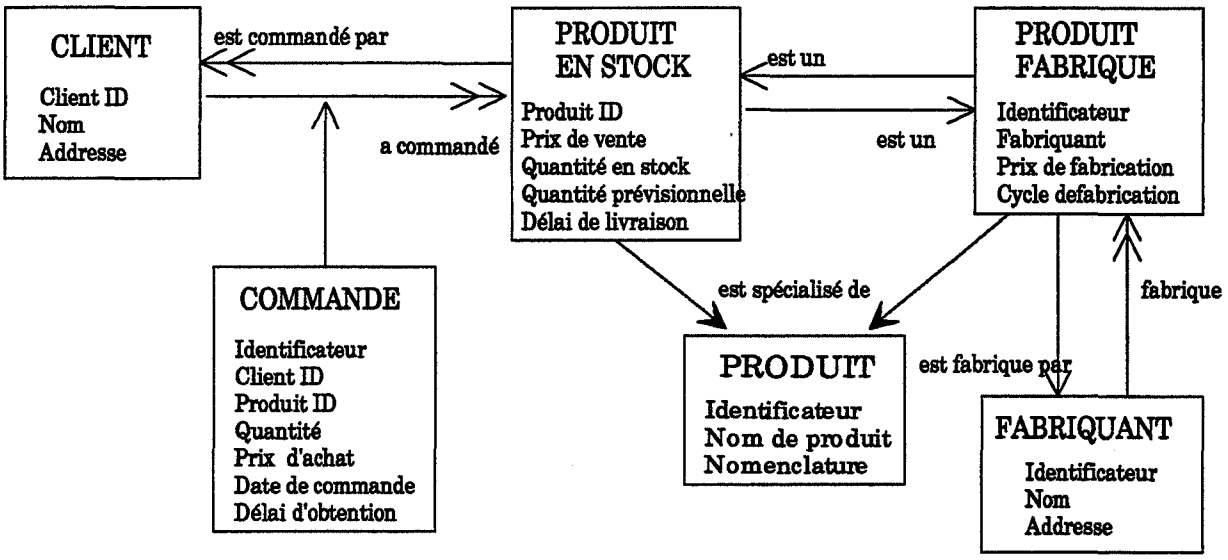


Figure 2-6 Modèle Entité-Association de Traitement de Commande

III.3 La Démarche de la Modélisation

La démarche est basée à la fois sur le cycle de vie des systèmes, les niveaux d'abstractions et le cycle des décisions qui doivent être prises. Elle peut être illustrée par le schéma de la figure suivante (figure 2-7).

Le schéma directeur couvre l'ensemble de l'organisme du système-entreprise. Son objectif est la planification à moyen terme du système d'information. A l'issue du schéma directeur opérationnel, le champ d'investigation a été découpé en domaines cohérents.

L'étude préalable doit donner aux responsables des éléments pour décider. Pour cela, elle doit notamment permettre d'évaluer: la faisabilité fonctionnelle et technique du système, la cohérence du planning et du budget de développement et de mise en place, ainsi que les coûts prévisionnels. L'étude préalable déroule le cycle d'abstraction sur les trois niveaux (conceptuel, organisationnel et physique).

L'étude détaillée permet de spécifier de façon détaillée et exhaustive la solution conceptuelle et organisationnelle retenue à l'issue de l'étude préalable. D'autre part, l'étude détaillée complète et valide la solution technique globale. Elle génère un dossier de spécifications fonctionnelles comportant: les événements et résultats traités, les processus de gestion mis en jeu, les procédures de traitement associées et les tâches.

L'étude technique est l'étape où l'on passe des spécifications détaillées et des contraintes qui constituent une expression de besoins, à la définition d'une solution en termes de traitements au niveau opérationnel et de données au niveau physique.

La réalisation couvre la production des logiciels et la mise en oeuvre; elle permet d'étudier l'ensemble des problèmes liés à l'utilisation efficace de nouvelles fonctions automatisées.

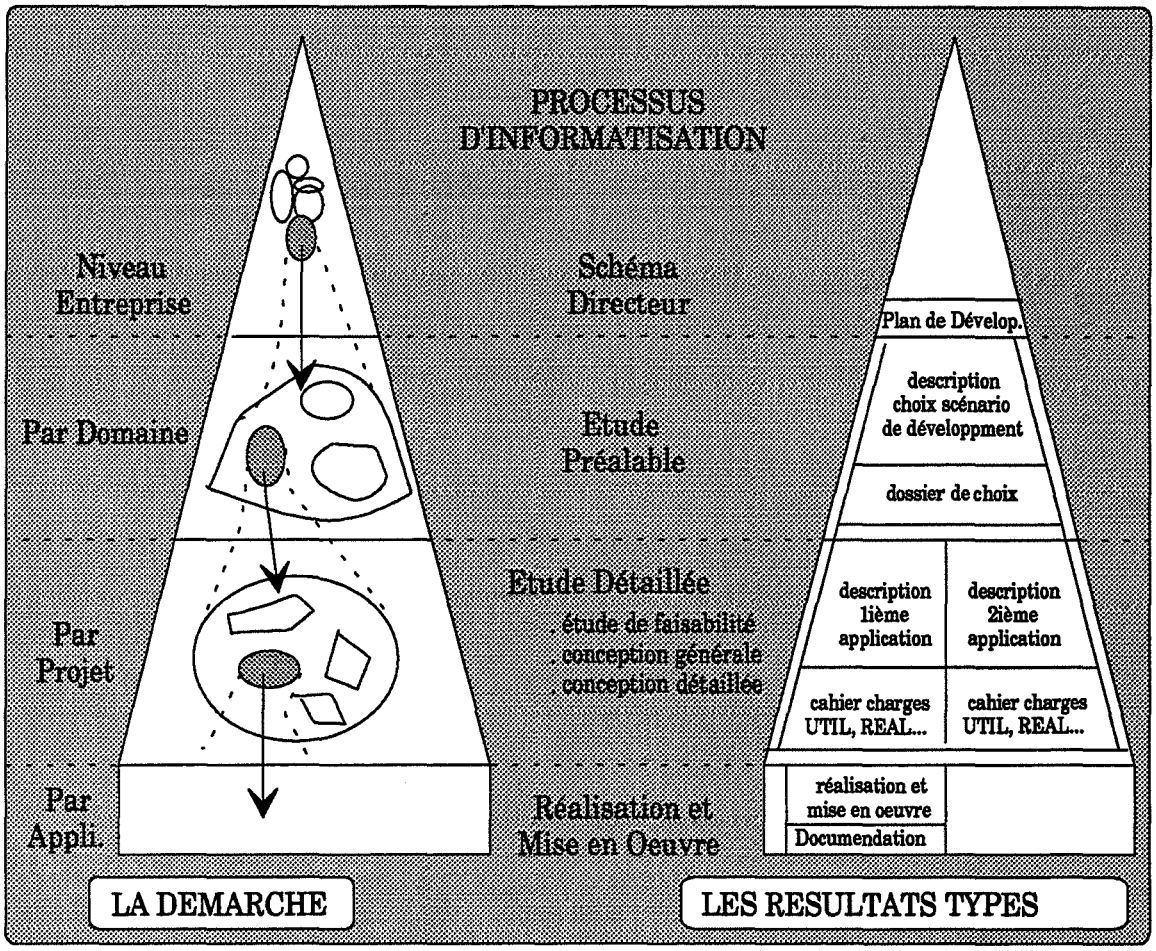


Figure 2-7 Démarche par Etapes de la Méthode MERISE

III.4 Conclusion sur la Méthode MERISE

La méthode MERISE est une démarche structurée qui s'applique à la conception de systèmes d'information. Cette démarche s'appuie sur des modèles de données et de traitements qui s'affinent au fur et à mesure de la progression du projet, partant de modèles conceptuels, pour aboutir, par l'intermédiaire de modèles logiques et organisationnels, aux modèles physiques selon la réalisation des programmes et la mise en place des bases de données entreprises.

Par rapport à la méthode SADT, MERISE est une méthode cohérente et assez complète. Il est vrai qu'elles ont plus de similitudes que de différences [QUANG 89], les plus importantes de ces différences sont les suivantes:

- La méthode SADT est avant tout une méthode d'analyse plutôt que de traitement, ce qui s'explique par la présence prédominante des actigrammes. Le tableau 2-2 montre les différents niveaux de traitements de SADT et de MERISE.

- La méthode de décomposition est différente: dans la méthode SADT, le choix et le nombre des différents niveaux d'abstraction sont du ressort de l'utilisateur; par contre la méthode MERISE intègre les différents niveaux d'abstraction et les fige.

- La méthode SADT est particulièrement bien adaptée aux étapes de spécification préliminaire et détaillée d'un système; la méthode MERISE privilégie un découpage sur les différentes catégories de décideurs (direction générale, utilisateurs, organisateurs, informaticiens, etc.).

<table>
<tr><th>méthode
niv.
d'abstraction</th><th>SADT</th><th>MERISE</th></tr>
<tr><td>CONCEPTUEL</td><td>LOGIQUE</td><td>CONCEPTUEL</td></tr>
<tr><td>LOGIQUE</td><td rowspan="2">PHYSIQUE</td><td>ORGANISATIONNEL</td></tr>
<tr><td>PHYSIQUE</td><td>PHYSIQUE</td></tr>
</table>

Tableau 2-2 Les Niveaux de Traitements des Méthodes MERISE et SADT

- La méthode SADT est une méthode pour communiquer des idées qui ne nécessite aucune connaissance informatique. Elle est facile à lire et à comprendre. La méthode MERISE est une méthode informatique. Elle est beaucoup plus complexe que la méthode SADT.

La méthode MERISE a été constamment améliorée au cours de ses dix années d'existence: la représentativité des modèles s'est affinée (vers les modèles objets [ROCHELD 93]) et les domaines d'emploi se sont étendus vers [PIERREVAL 90]: les systèmes de gestion de production, les systèmes de communication, la planification et le suivi des projets et l'étude

technique. Il existe quand même des limites à ces extensions dans l'informatique à caractère technique [TARDIEU 89].

- Le cycle de vie des systèmes est différent en informatique de gestion et en informatique technique.

- Les modèles de traitements et de flux d'information de l'informatique de gestion sont inadaptés à la représentation des traitements hautement parallèles que l'on trouve en informatique technique.

- La conception en informatique de gestion débouche sur des environnements base de données et/ou base de connaissances qui ne sont pas ceux que l'on rencontre en informatique technique.

Une approche nouvelle est donc nécessaire pour aborder correctement le domaine de l'informatique technique. En ce qui concerne les systèmes de production, deux modèles (méthodes) méritent d'être présentés dans la section suivante: modèles GRAI et CIMOSA.

IV Les Méthodes GRAI et CIMOSA

Pendant très longtemps, l'automatisation des systèmes de gestion de production s'est faite sur le modèle des méthodes informatiques (SADT, MERISE, SA/SD) ou de certains modèles de réseaux (réseau de Pétri, réseaux de files d'attente) décrivant le système physique de production. Or un système de gestion de production comprend trois sous-systèmes: un sous-système physique, un sous-système d'information et un sous-système de décision. De par leur nature et leur diversité, ces systèmes relèvent de la plus grande complexité [MOIGNE 90].

Comprendre le fonctionnement des systèmes du monde réel nécessite des méthodes et outils appropriés. L'insuffisance des méthodes informatiques "pour traiter" le système de gestion de production dans sa totalité a contribué au développement de méthodes et outils d'analyse propres à la gestion de production [ADAMA 84]. La figure 2-8 montre la spécificité des principales méthodes d'analyse/conception, par rapport au système de gestion de production et à ses sous-systèmes. Nous ne présentons ici que deux méthodes qui sont, à notre connaissance, les plus utilisées en France: la méthode GRAI et la méthode CIMOSA.

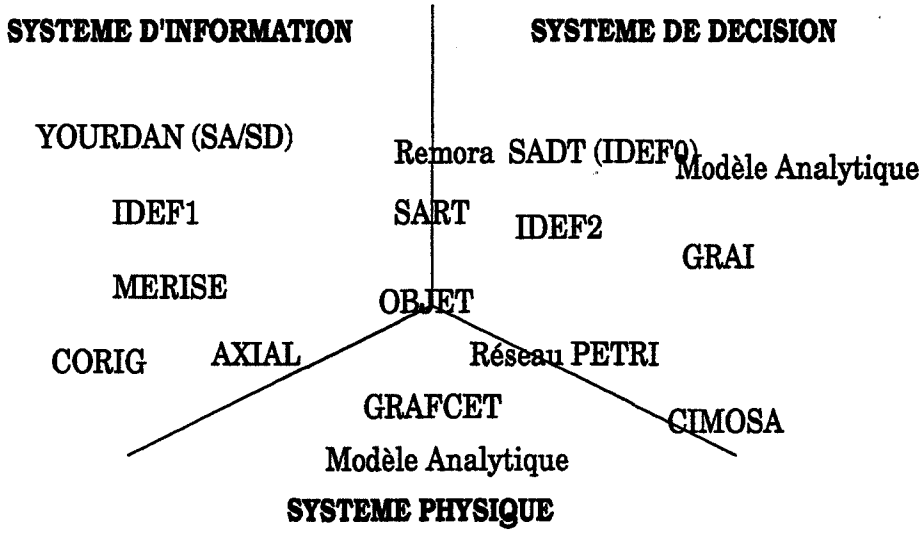


Figure 2-8 Les Méthodes d'Analyse par Rapport au Système de Gestion de Production

IV.1 La Méthode GRAI

La méthode GRAI (Graphes à Résultats et Activités Inter-reliés) [ADAMA 84] s'appuie sur deux descriptions conceptuelles du système de gestion de production (SGP), élaborées à partir des théories sur les systèmes hiérarchisés et des systèmes d'organisation. Cette méthode comprend:

- un *modèle conceptuel global* élaboré d'une part à partir des théories des systèmes d'organisation et d'autre part à partir des systèmes hiérarchisés.

- un *modèle conceptuel du centre de décision* qui est une approche "micro" du modèle conceptuel global.

- deux outils de représentation graphique des deux modèles: *la grille et les réseaux GRAI.*

Le modèle conceptuel du système de gestion de production (figure 2-9) exprime la décomposition des systèmes de production, introduite plus haut, en trois composantes: le sous-système physique, le sous-système d'information et le sous-système de décision. Le système de gestion de production se définit comme la réunion du système de décision et du système d'information.

Le modèle conceptuel du centre de décision exprime plus explicitement le mécanisme de fonctionnement d'un centre de décision. Un *centre de décision* (figure 2-10) est un ensemble

d'*activités de décision* (ou activités décisionnelles) que l'on isole des autres activités à partir des critères suivants.

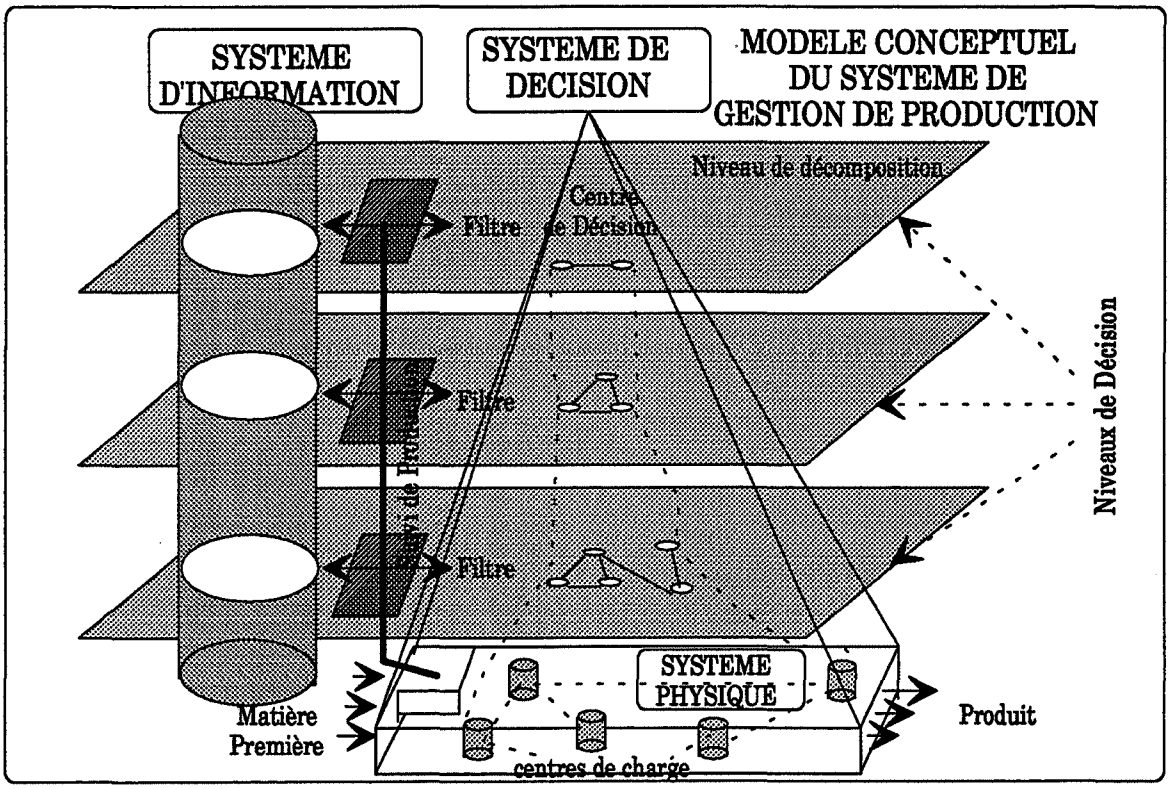


Figure 2-9 Modèle Conceptuel Global du Système de Gestion de Production (SGP)

- Les *critères fonctionnels* qui permettent de distinguer les activités selon les fonctions de base auxquelles elles appartiennent. Le plus souvent ces fonctions sont: planifier, approvisionner, gérer les ressources, fabriquer, contrôler et livrer.

- Les *critères temporels* exprimés en termes d'*horizon* de temps concerné par une prise de décision et de *période* relative à l'intervalle de temps séparant la remise en cause des décisions. Ces critères permettent de placer les activités, et donc les centres de décision, sur des *niveaux de décision* caractérisés par un couple (horizon, période).

La méthode GRAI dispose de deux outils graphiques pour représenter les deux modèles précédents: la grille et le réseau GRAI. *La grille GRAI* se présente comme un tableau dont:

- les colonnes sont les fonctions élémentaires du système de gestion de production,

- les lignes, les différents horizons de prise de décision; chaque horizon étant associé à une période de remise en cause de la décision.

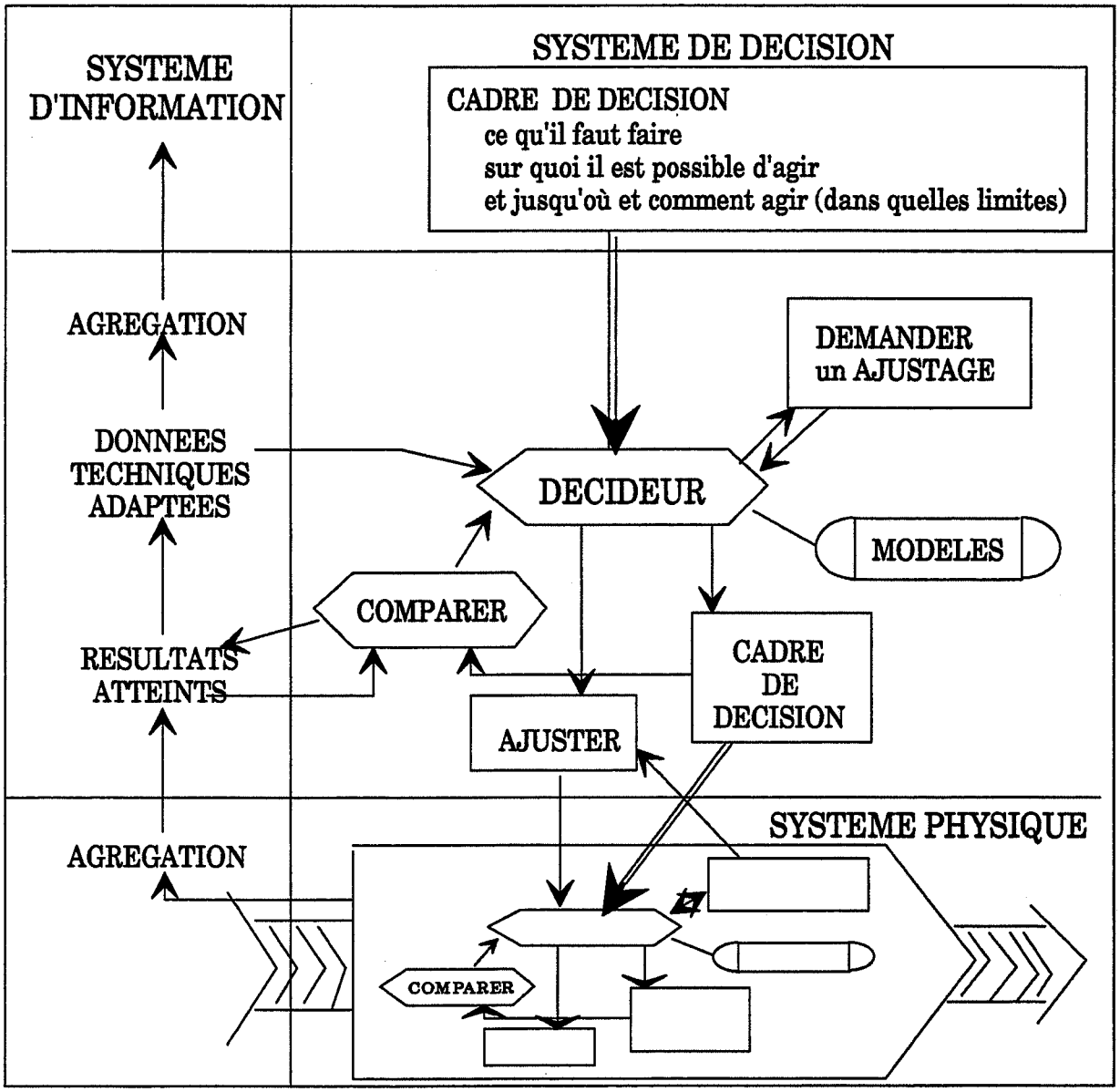


Figure 2-10 Modèle Conceptuel d'un Centre de Décision

Cette décomposition en fonctions et niveaux de décision définit une matrice, représentée graphiquement par une grille dont les cases sont les centres de décision. La grille couvre la totalité de la structure du système sans entrer dans le détail, pour cela on peut considérer que la grille constitue une macro-modélisation du système de gestion de production (SGP).

Chaque centre de décision, déterminé par la grille, peut être analysé par *les réseaux GRAI*, activité par activité. Deux types d'activités sont représentés dans les réseaux: *les activités d'exécution* et *les activités de décision.*

- En phase d'analyse, ces réseaux expriment les activités exécutées dans le SGP, et servent à lever l'ambiguïté sur l'exécution de certaines tâches ou sur les mécanismes de prise de décision. Ils permettent d'étudier la synchronisation des activités et le fonctionnement du système en régime perturbé. Cette étude s'effectue au niveau même d'un centre de décision, mais elle concerne également les relations entre différents centres.

- En phase de conception, les macro-GRAI permettent une meilleure compréhension de l'architecture proposée par la grille de conception. Notamment pour la description du contenu des liens décisionnels. Les réseaux permettent de définir les spécifications fonctionnelles du système à réaliser qui serviront, entre autres, à l'élaboration du cahier des charges de nouveaux systèmes et éventuellement à la recherche d'un progiciel de GPAO.

La méthode GRAI fournit également une démarche d'application en permettant à différents intervenants d'y participer. La figure 2-11 schématise la méthodologie d'application. Bien que la méthode fournisse des outils d'aide, l'expérience d'un spécialiste reste prépondérante [PIERREVAL 90]. Cette dernière est nécessaire à la fois pour la pratique de la méthode GRAI (notamment pour la construction des grilles et des réseaux) et pour les connaissances en gestion de production qui restent indispensables lors des phases d'expression de dysfonctionnements et de conception.

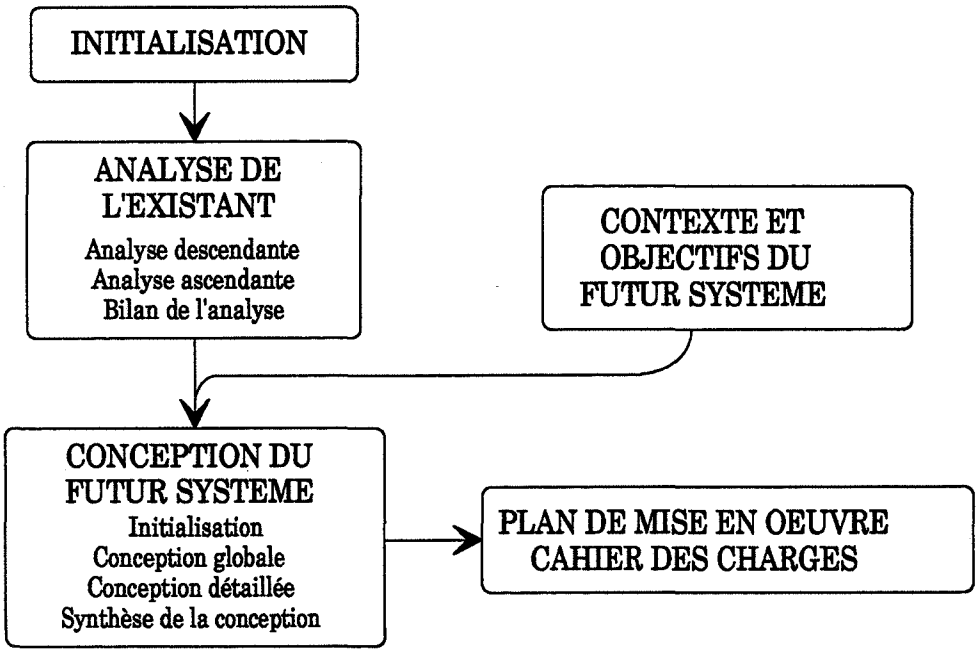


Figure 2-11 Méthodologie d'Application de la Méthode GRAI

Des extensions de la méthode sont en cours [DOUMEING et al. 90]: l'intégration de l'évaluation économique (méthode ECO-GRAI), la définition d'approches méthodologiques du

système physique et la définition de "modèles de références" destinés à faciliter le diagnostic et la conception (modèle NBS). Basés sur cette méthode, différents projets européens sont en-cours [DOUMEING 91]: l'architecture MMCS (Manufacturing Management Control System), l'architecture GIM (GRAI Integrated Method), le projet IMPACS (Integrated Manufacturing Planning And Control System), etc. Nous présentons ici une méthode européenne: la *méthode CIMOSA* qui est une architecture de système ouvert pour la productique.

IV.2 La Méthode CIMOSA

CIMOSA (Open Systems Architecture for Computer Integrated Manufacturing) est une architecture de systèmes ouverts pour la productique développée par le Consortium AMICE dans le cadre du programme ESPRIT (Projets No. 688, 5288 et 7110). L'architecture est articulée autour de trois composants fondamentaux (figure 2-12) [AMICE 89].

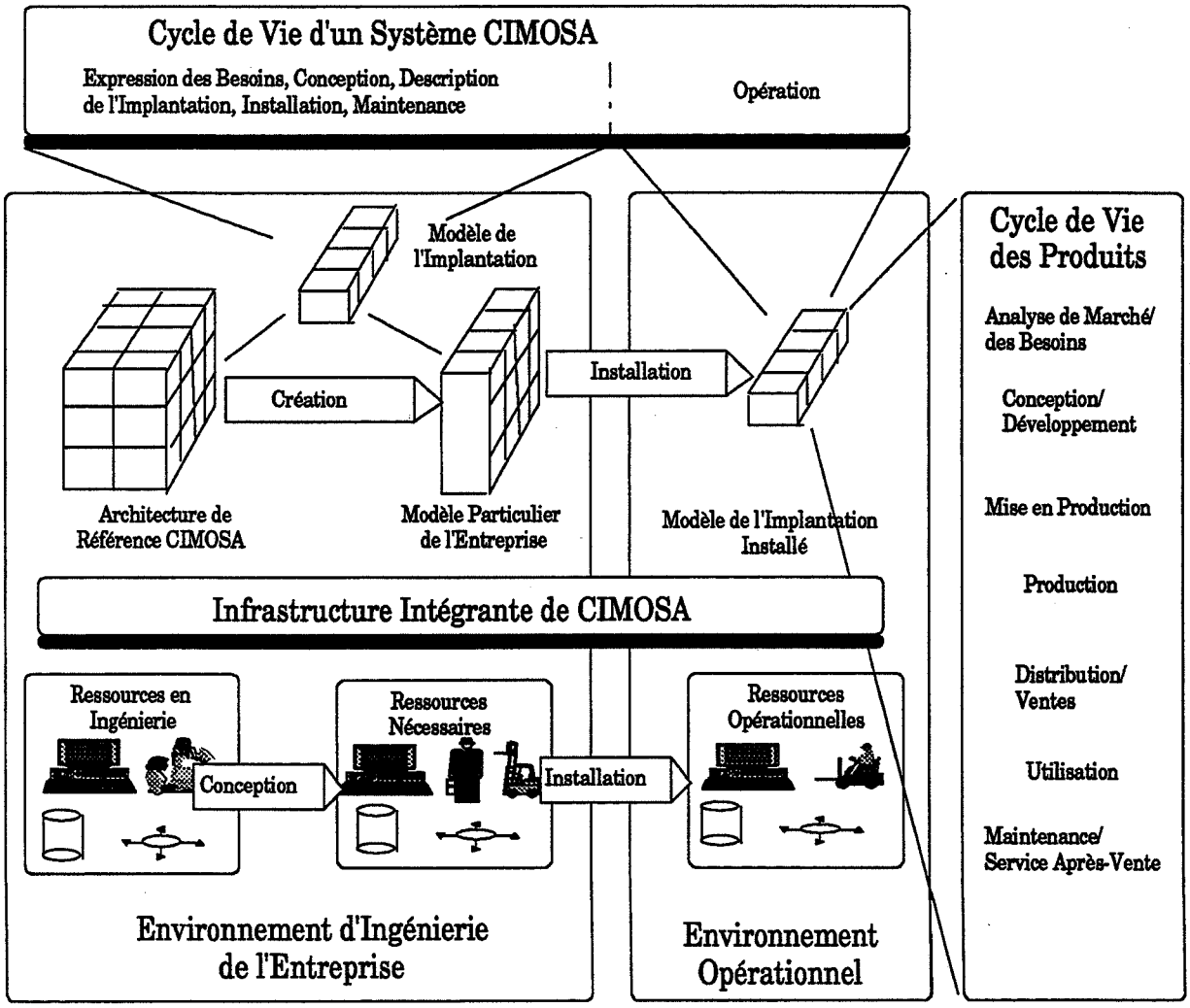


Figure 2-12 L'Architecture CIMOSA

- Un cadre de modélisation d'entreprise (incluant une Architecture de Référence qui fournit une intégration conceptuelle par unification sémantique);

- Une infrastructure intégrante (permettant l'intégration physique et l'intégration des applications);

- Un cadre méthodologique (couvrant le cycle de vie du système de production et assurant la cohérence de l'ensemble).

IV.2.1 Le Cadre de Modélisation de CIMOSA

Le but du cadre de modélisation de CIMOSA ou du cube CIMOSA (en anglais, Modeling Framework) est de fournir un cadre conceptuel, une méthode et des outils de modélisation pour assister l'utilisateur dans le développement du modèle particulier de son entreprise. Ce cadre est donc composé de deux parties essentielles (figure 2-13): l'une appelée *Architecture de Référence* (fournie par AMICE) et une autre appelée *Architecture Particulière* (propre à l'entreprise). Ce cadre s'articule autour de trois axes de modélisation orthogonaux:

- l'*axe de généricité* qui suggère de construire le *modèle particulier* de l'entreprise à partir de *modèles partiels*, s'il en existe, eux-mêmes exprimés en termes de *concepts de base génériques* (Generic Building Blocks) ou de leurs types et sous-types. Les concepts de base génériques et les *types de concepts de base* (Building Block Types) forment le "langage de modélisation de CIMOSA". Seul le Consortium AMICE ou les organes de normalisation sont susceptibles de modifier ou enrichir ce langage;

- l'*axe des modèles* qui invite à modéliser d'abord les besoins précis de l'entreprise (définition des objectifs, élaboration du cahier des charges), puis à concevoir les spécifications du système CIM (analyse conceptuelle, analyse détaillée, conception du système d'information, évaluation des performances, choix des ressources) et enfin à décrire l'implantation (ressources installées, distribution des fonctions, distribution des informations, gestion des aléas, etc.);

- l'*axe des vues* qui propose de gérer le modèle intégré (conception, manipulation, accès) suivant quatre points de vue (fonctions, informations, ressources et organisation) pour faire face à la complexité du système et de son modèle.

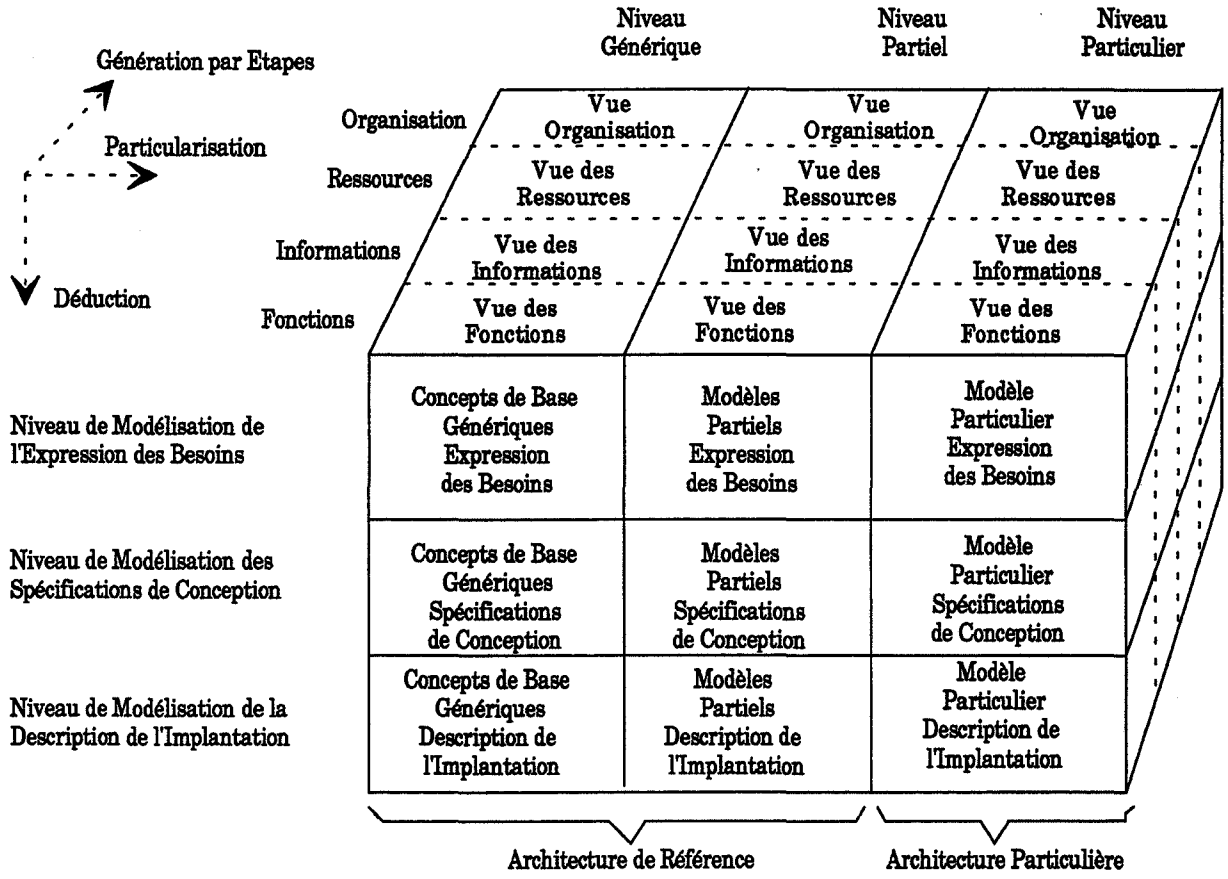


Figure 2-13 Le Cadre de Modélisation de CIMOSA

Du point de vue fonctionnel, une entreprise est vue dans CIMOSA comme un ensemble de *domaines* (parties disjointes de l'entreprise) formés de *processus* qui interagissent entre eux (traitement des commandes-clients, planification de la production, traitement de nomenclature, etc.). Ces processus d'entreprise sont déclenchés par des *événements* provenant du modèle lui-même ou du monde extérieur (pannes de machines, ordres de gestion, etc.). Un processus est formé de sous-processus et d'*activités*. C'est en fait une chaîne d'activités spécifiée au moyen de *règles procédurales* qui décrivent le comportement de l'entreprise en fonction des événements reçus et de l'état du système. Une activité (figure 2-14) décrit une tâche réalisée dans l'entreprise au moyen de ressources (entrée ressource) au cours du temps et consistant à transformer des intrants (entrée fonctionnelle) en extrants (sortie fonctionnelle) sous certaines contraintes (entrée de contrôle) et produisant des informations de sortie (sortie de contrôle et sortie ressource). Les activités décrivent la fonctionnalité de l'entreprise. Une activité consiste à exécuter une séquence d'*opérations* suivant une logique donnée. Les opérations représentent le plus bas niveau de granularité dans la vue des fonctions. Toute activité produit en sortie un *état de fin* indiquant comment s'est terminée l'activité. Cet état de fin est utilisé par les règles procédurales des processus pour enchaîner les activités au cours du temps.

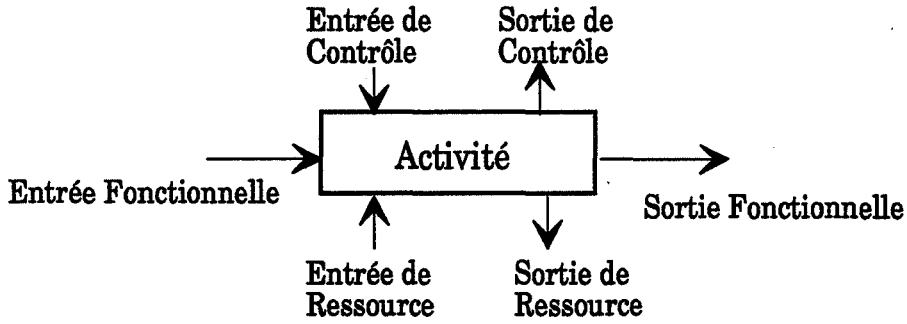


Figure 2-14 Diagramme de l'Activité

Du point de vue informationnel, la plupart des entrées/sorties des activités sont des *vues d'objet*. Une vue d'objet est une *manifestation* d'un objet tel qu'il est perçu par un utilisateur ou une application. Elle peut être de nature physique ou de nature informationnelle. Dans le cas de vue d'objet de nature informationnelle, la vue d'objet est définie par un nom et un type de données (types de base usuels en informatique). Derrière la description des vues d'objets se cache un objet d'entreprise qui est une représentation abstraite (ou informatique) d'une entité réelle de l'entreprise et qui est définie par tous les éléments d'information relatifs à cet objet complexe de l'entreprise (fait d'autres objets) dont on peut produire des contraintes de copies (ou occurrences) mais dont on ne perçoit que des vues d'objet.

Du point de vue de la gestion des ressources, les activités ont besoin de *ressources* pour leur exécution. CIMOSA distingue deux classes de ressources: les ressources actives, appelées *entités fonctionnelles* et les ressources passives (outils, palettes, etc.), appelées *composants*. Les entités fonctionnelles jouent un rôle important dans un système CIM car ce sont celles qui exécutent les opérations des activités de la vue des fonctions. On distingue dans CIMOSA trois grandes classes d'entités fonctionnelles: les *machines* (machines outils, systèmes de transport, robots, etc.), les *applications* (systèmes de CAO, de M.R.P., logiciels d'ordonnancement, etc.) et les *hommes* (opérateurs, décideurs, gestionnaires, concepteurs...). Ces entités fonctionnelles sont connectées à l'architecture par les services de présentation de l'infrastructure intégrante.

Du point de vue organisationnel, CIMOSA fournit des concepts de base génériques pour décrire les *responsabilités* dans l'entreprise, les niveaux organisationnels (*unité d'organisation* et *cellule d'organisation*) et les cadres de décision.

IV.2.2 L'Infrastructure Intégrante de CIMOSA

L'infrastructure intégrante de CIMOSA (en anglais Integrating Infrastructure ou IIS) est constituée de cinq grands groupes de services, appelés *entités* (figure 2-15). Leur but est de fournir un ensemble commun de services informatiques de base répartis formant une plate-

forme d'intégration unifiée pour les différents composants du système CIM (fonctions, centres d'informations, ressources et entités d'organisation) et devant servir de support à l'exécution des activités de l'entreprise. En d'autres termes, l'IIS vise à transformer un environnement hétérogène en un environnement homogène et donc plus facilement gérable.

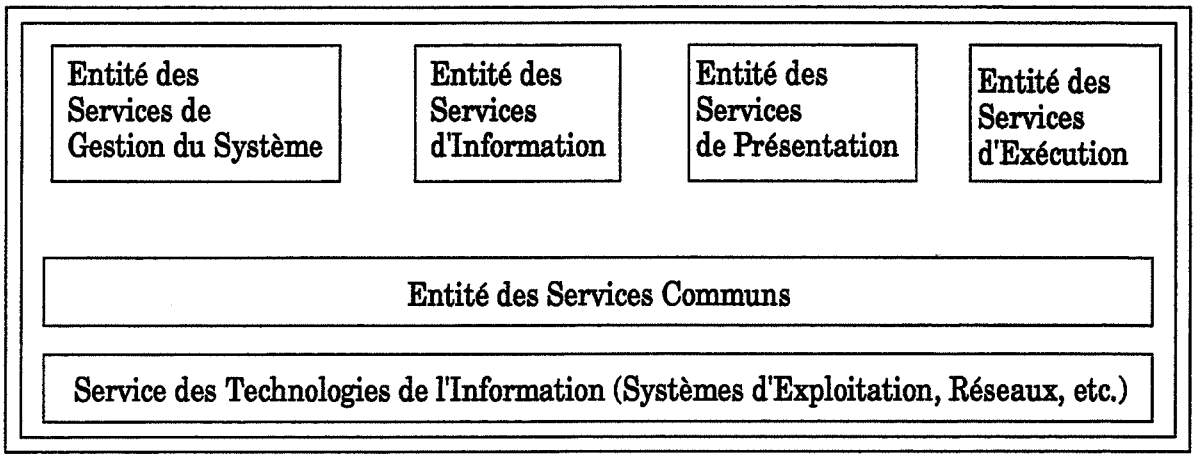


Figure 2-15 Services de l'Infrastructure Intégrante

IV.2.3 La Méthodologie de Développement

CIMOSA offre une méthodologie d'intervention en accompagnément du cycle de vie du système CIM. Les phases du cycle de vie du système sont illustrées dans la figure 2-16. Il est apparu dans le projet AMICE que ce ne sont pas les trois premières phases qui sont les plus difficiles à réaliser comme on aurait pu le croire, mais la phase de maintenance/modification. On peut toujours réaliser un système à base de technologie de l'information d'une façon ou d'une autre, mais changer dynamiquement un tel modèle en cours d'exploitation reste un problème entier pour lequel il n'y a pas de solution à ce jour [GACHES et al. 93].

CIMOSA a pour but de fournir un support tout au long du cycle de vie du système CIM, en particulier:

- pour la définition précise des objectifs de l'entreprise et des stratégies manufacturières;
- pour permettre de configurer et de gérer l'exploitation du système en réponse à ses objectifs;
- pour permettre de gérer le système dans un contexte en changement perpétuel.

Les avantages de CIMOSA concernent, entre autres choses, une modélisation cohérente de l'entreprise depuis l'expression précise des besoins jusqu'à une description conforme de

l'implantation, une véritable intégration des divers composants du système CIM et l'interopérabilité avec des composants provenant de vendeurs différents.

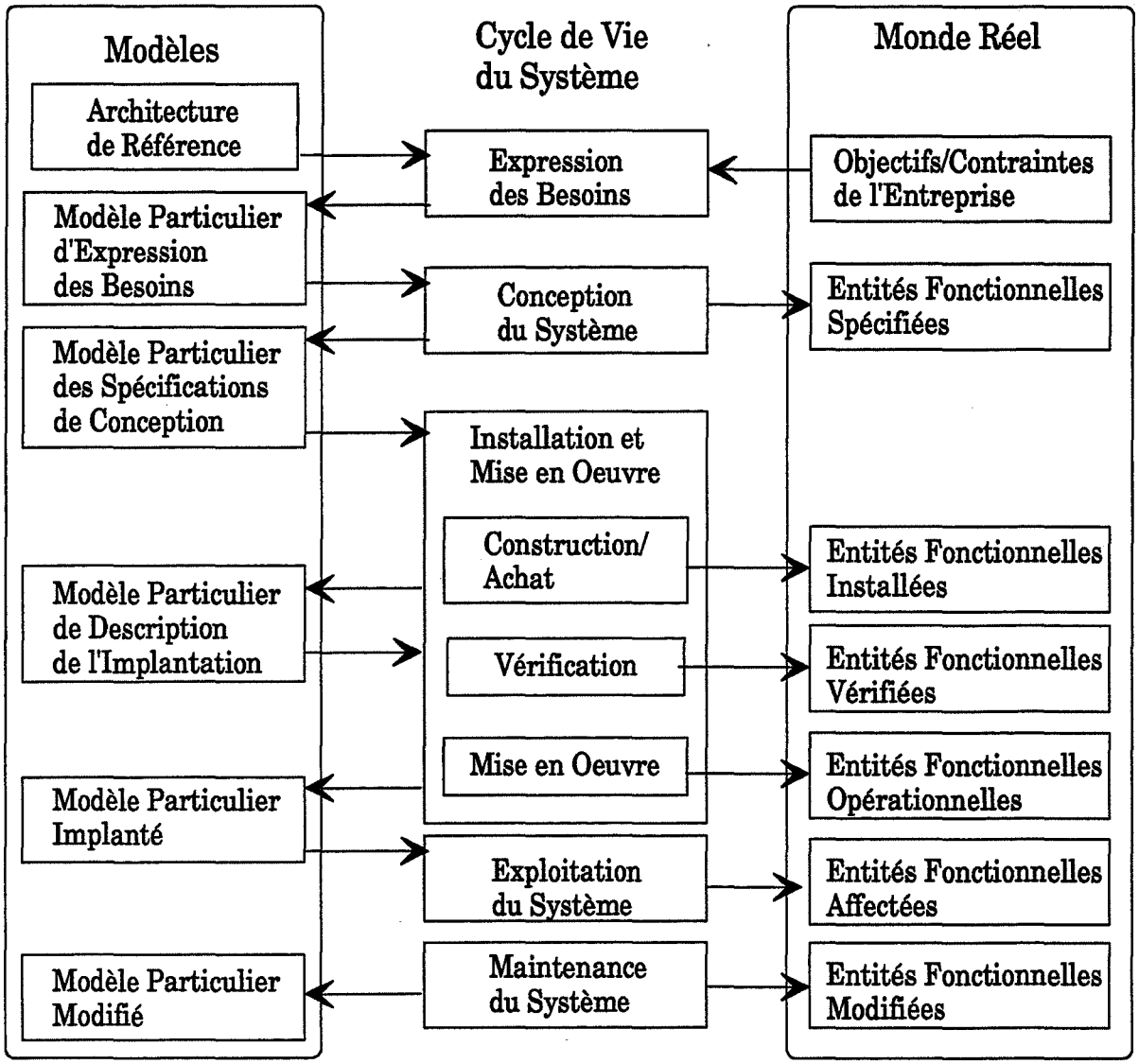


Figure 2-16 Cycle de Vie d'un Système CIM

IV.3 Conclusion sur les Méthodes GRAI et CIMOSA

Pour conclure sur ces deux méthodes relatives à la gestion des systèmes de production, nous pouvons nous appuyer sur leurs champs d'application.

La méthode GRAI s'applique essentiellement dans le domaine de la gestion de production en utilisant une connaissance a priori du système de décision et du système d'information.

La méthode CIMOSA fournit une architecture des systèmes ouverts de productique qui permet de modéliser intégralement une entreprise en se basant sur le cadre de modélisation et sur l'infrastructure intégrante de CIMOSA (figure 2-17). Par exemple, pour intégrer deux systèmes hétérogènes (Système A et Système B), l'infrastructure de CIMOSA permet de relier physiquement les deux systèmes et d'assurer les échanges de matières (flux physique) et d'information (flux d'informations). Le cadre de modélisation sert de référentiel et réalise l'unification des concepts entre les systèmes. Les deux systèmes "parlent" le même langage et ont une vision commune du système global, même si intérieurement ils utilisent des langages spécialisés et des technologies différentes.

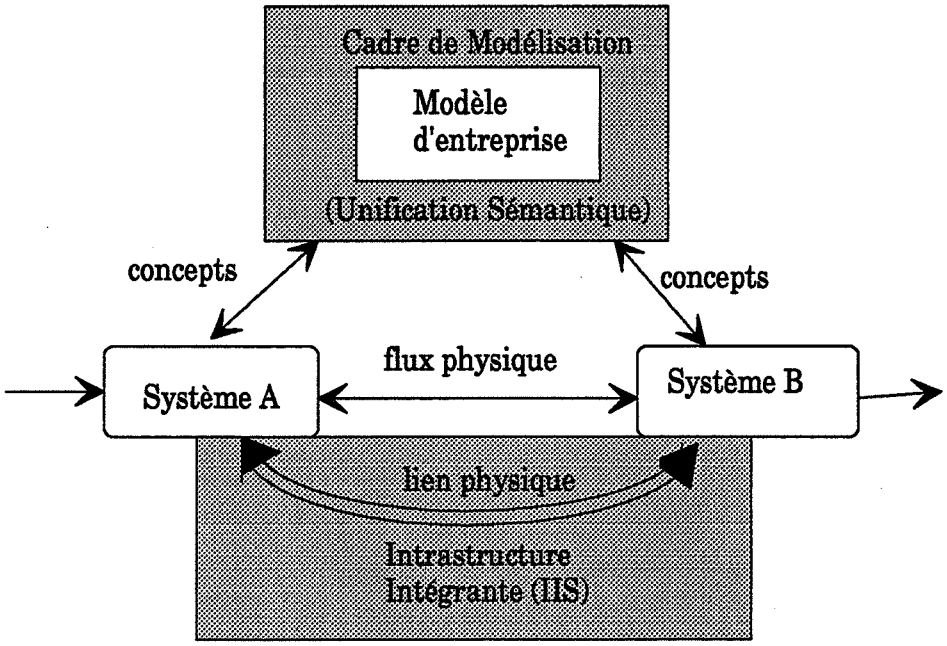


Figure 2-17 Intégration des Systèmes CIM par la Méthode CIMOSA

Evidemment, des améliorations de ces deux méthodes sont envisagées. Dans la méthode GRAI, les données ne sont pas traitées en parallèle avec les traitements. Elle s'intéresse plutôt à la modélisation de la structure décisionnelle. Le fait de présenter une structure théorique du système de production (également dans le modèle CIMOSA) facilite l'approche mais peut dans certains cas être restrictif. En ce qui concerne la méthode CIMOSA, elle offre un cadre innovateur et intéressant pour l'intégration des systèmes manufacturiers. Cependant, un effort important reste encore nécessaire à ce jour pour faire de CIMOSA une réalité dans le monde industriel (cohérence des informations dans les différents modèles de CIMOSA, l'évolution du modèle en suivant la modification du système réel, etc.).

IV.4 Conclusion sur les Méthodes d'Analyse et de Conception

Dans le cadre de la modélisation des systèmes de production, nous distinguons deux types de modèles: les *modèles de structure* et les *modèles de simulation*. Les méthodes ou les modèles présentés précédemment correspondent aux modèles de structure. Ils définissent les concepts de base, les éléments et les relations entre ces éléments d'un point de vue statique. Les modèles de simulation reprennent les concepts définis dans les modèles de structure et introduisent le facteur "temps": ils prennent en compte la dynamique du système.

Nous avons vu que toutes les méthodes précédentes définissent un ensemble de concepts, d'outils et d'étapes à suivre pour construire un modèle structuré d'entreprise qui représentent les éléments invariants de l'ensemble du système productique. Cependant, un système de production à modéliser est par nature complexe, et ce d'autant plus qu'il s'agit d'un système stochastique soumis à perturbations et aléas. Un modèle de simulation est nécessaire pour valider ou évaluer les performances du système étudié.

V La Simulation

La simulation est une technique de modélisation qui consiste à reproduire le comportement dynamique d'un système sur ordinateur afin de mieux le connaître, de mieux maîtriser son évolution dans le temps dans un environnement donné, et d'évaluer ses performances.

La modélisation, et plus particulièrement la conceptualisation du problème est l'étape la plus importante d'un processus de simulation. La difficulté est de traduire un problème qui s'exprime en termes réels et concrets au moyen d'un vocabulaire industriel en un modèle constitué de primitives relativement abstraites (vocabulaire de simulation). Nous nous intéressons dans ce mémoire à la modélisation des systèmes à événements discrets comme c'est le cas de la plupart des systèmes manufacturiers.

V.1 Simulation à Evénements Discrets

Simuler un système, c'est chercher à reproduire son évolution dans le temps, dans un environnement donné; cette évolution est caractérisée par celle de variables représentatives du système appelées *variables d'état*.

Dans une simulation à événements discrets, les variables d'état que l'on désire connaître à tout instant sont discrètes [LEROUDIER 80]. L'ensemble des valeurs que ces variables peuvent prendre constitue l'*espace d'état* du système. Il s'ensuit que l'espace d'état est *dénombrable* ou

fini. Du fait de sa définition, on remarque que chaque *changement d'état* ou *événement* se produit d'une manière discrète dans le temps à des instants que l'on appelle des *dates d'événement*. En général les événements sont contingents, c'est-à-dire que leur occurrence dépend de l'évolution du système et donc d'autres événements antérieurs. Néanmoins, il peut exister des événements dits déterminés qui ont lieu à certains instants prédéfinis par une horloge du système.

Pendant tout intervalle de temps où l'état d'un objet du système ne change pas on dit que l'objet est engagé dans une certaine *activité*. Un événement est donc un changement d'état qui initialise une activité qui n'était pas en cours auparavant. L'état du système global à un instant donné peut être caractérisé par l'ensemble des activités en cours. Le recensement de ces activités peut être effectué à partir des valeurs des variables (attributs de tous les objets) du système et réciproquement.

D'après les définitions ci-dessus, on constate la dualité entre événement et activité: toute activité est limitée à son début et à sa fin (début d'une autre activité) par un événement. Tout événement marque le début d'une activité (qui peut être de durée nulle).

Quand, dans le modèle, on rencontre souvent des séquences d'événements ou d'activités similaires pour un type d'objet, on peut définir ce qu'on appelle un *processus*. Un processus est la succession d'un nombre fini d'états d'un objet ou de façon équivalente la succession d'une ou plusieurs activités qui concernent cet objet. Il permet de faciliter la description d'un système.

Pour que la simulation soit possible, il faut être capable de décrire les changements d'état ou événements par des *algorithmes*. Pour chaque événement on doit définir dans quelles activités vont s'engager les différents objets libérés par cet événement (fin de l'activité précédente). On définit ainsi des contraintes de précédence entre les différentes activités. Les traits caractéristiques de la dynamique de ces modèles sont les suivants [BEL et al. 85]:

- *Parallélisme des activités et phénomènes de synchronisation*. Plusieurs activités peuvent avoir lieu en même temps. Une activité ne peut commencer que quand les objets concernés sont libres.
- *Non déterminisme* au sens où certains changements d'état ne peuvent se faire qu'en surajoutant des règles de décision au modèle.

En ce qui concerne les systèmes de production, les changements d'état sont décrits par deux types de règles. D'une part ils sont dictés par des contraintes technologiques (gammes d'usinage, par exemple) qui ne peuvent pas être modifiées dans le modèle, ce sont des *règles*

opératoires. D'autre part ils sont déterminés par des *règles de gestion et de pilotage* qui ne sont pas connues a priori. Le choix des règles va déterminer les performances du système grâce à un meilleur contrôle du flux de production.

V.2 Modélisation de Simulation à Evénements Discrets

Les événements arrivant d'une manière discrète, ils peuvent être ordonnancés dans le temps (ordre chronologique) et donc simulés sur une machine séquentielle l'un après l'autre. Le parallélisme qui peut apparaître à l'utilisateur au niveau du fonctionnement logique du système, n'existe pas, en réalité, au niveau des changements d'état. Par la manière de gérer le temps dans une simulation à événements discrets, on distingue les simulations dirigées *par une horloge* et les simulations dirigées *par événements* [LEROUDIER 80].

- *Simulation dirigée par une horloge*. On définit une unité de temps appropriée au problème et on dispose d'une horloge centrale qui progresse par pas d'une unité de temps. A chaque incrémentation de l'horloge, on explore la liste des événements pour voir si l'un d'eux apparaît à cette date ou si les conditions pour commencer ou terminer une activité sont remplies. Ce mécanisme est souvent utilisé dans l'approche par activités. L'inconvénient de cette approche vient du fait qu'il faut tester toutes les activités ou ordonnancer tous les événements à chaque pas d'avance du temps et qu'elle risque ainsi de demander un trop grand temps de calcul.

- *Simulation dirigée par événements*. Les seuls temps accessibles lors de la simulation sont des dates d'événements et l'incrémentation du temps se fait d'une date d'événement à l'autre. Le temps de la simulation est alors géré à partir d'un *échéancier* que l'on peut présenter conceptuellement comme une liste linéaire d'événements ou de processus ([PRITSKER 86], [PEGDEN et al. 90]. La tête de la liste est l'événement (processus) courant (présent) et la fin de la liste l'événement (processus) le plus éloigné dans le futur. Il faut remarquer qu'à part l'événement (processus) courant, les autres événements (processus) sont purement *potentiels* car leurs apparitions peuvent être remises en cause lors du traitement des événements (processus) qui les précèdent.

Nous nous intéressons dans ce mémoire aux simulations dirigées par événements. Pour pouvoir réaliser une simulation, en plus de l'échéancier, il faut des procédures qui permettent de l'entretenir et de le manipuler. L'ensemble de ces programmes et de ces structures de données s'appelle un *noyau de synchronisation (scheduler)*. Un tel noyau de synchronisation existe sous une forme ou sous une autre dans toute simulation à événements discrets et en particulier dans

tout langage de simulation (SIMULA, SIMSCRIPT, SIMAN, SLAM, GPSS, etc.). Il existe deux approches pour réaliser ce mécanisme.

- Le noyau de synchronisation fournit des fonctions élémentaires permettant d'ordonnancer les événements dans le temps. Lorsque l'événement considéré arrivera en tête de l'échéancier, on exécutera alors le sous-programme associé à l'événement qui décrit le changement d'état considéré dans le système (une activité). Bien entendu, ce sous-programme peut ordonnancer l'événement qui a provoqué son appel ou d'autres événements. On dit que la simulation est basée sur la *notion d'événement*.

- Chaque activité de la simulation qui demeure toujours la description d'un changement d'état, donc d'un événement, est vue à un niveau logique comme l'activité d'un processus. Le modèle simulé est alors décrit comme un ensemble de processus progressant parallèlement dans le temps. Chaque processus représente une ***entité active*** (*un objet actif*) qui peut évoluer dans le temps au contraire des autres entités de la simulation qui demeurent passives. Du fait que les différentes entités sont exécutées en parallèles, elles ne peuvent plus être décrites par des sous-programmes (elles seraient alors purement séquentielles) mais elles le sont par des *coroutines* qui leur assurent une exécution en quasi-parallélisme que nous détaillerons dans le chapitre V. Le noyau de synchronisation fournit alors des ordres du type "activer le processus X à heure absolue T". Dans une telle mise en oeuvre, on dit que la simulation est basée sur la *notion de processus*. Cette approche permet une dualité dans la description du système à simuler en permettant un échange entre *entités passives* et *entités actives* (processus). Cette dualité est intéressante et permet de faire un choix sur le coût et les performances de la simulation projetée. Ce qui coûte dans une telle simulation, ce sont, bien entendu, les entités actives (processus); on visera donc à faire un choix qui en minimise le nombre.

V.3 Modélisation des Systèmes à Evénements Discrets.

Basé sur les concepts et les mécanismes de simulation à événements discrets, on distingue plusieurs approches pour modéliser les systèmes à événements discrets [CAVILLE et al. 87].

V.3.1 Approche par événements

Dans cette approche, on commence par répertorier tous les événements ou changements d'état susceptibles d'être rencontrés au cours de l'évolution du système. Puis on modélise la logique de changements d'état sous une forme d'algorithmes en définissant, pour chaque type d'événement, les conditions sur l'état conduisant à l'occurrence de l'événement ainsi que les

changements d'état correspondants. La simulation du système est obtenue par l'exécution des logiques de changements d'état associée à chaque événement à la date à laquelle celui-ci se produit.

V.3.2 Approche par cycle d'activités

Au lieu de répertorier des événements, dans cette approche on répertorie les types d'activités. La logique de changement d'état se fait alors en précisant les conditions nécessaires au début et à la fin d'une activité. Le déroulement de la simulation se fait à l'aide d'une horloge. A chaque pas, on teste, pour chaque activité, si les conditions nécessaires à son début ou à sa fin sont remplies.

V.3.3 Approche par processus

Dans cette approche, la logique de changement d'état est relative à une séquence d'événements prédéterminés ou processus. On choisira comme processus des séquences d'événements parfaitement déterminés pour lesquels la logique de changement d'état sera toujours la même. La modélisation d'un système nécessite donc:

- la définition des différents processus composant le système, et pour chacun d'eux, la logique de changement d'état décrivant le cheminement, événement par événement, de l'entrée à la sortie du processus;
- la connexion des différents processus et la spécification de leurs interactions. Ceci conduit à la description complète du système.

Cette approche combine la simplicité de la description de l'approche par cycle d'activités et l'efficacité de l'approche par événements. La plupart des langages de simulation, SLAM, SIMAN, QNAP2, etc. proposent des processus pré-programmés sous forme de primitives standard facilitant la modélisation. Par contre, la modélisation par cette approche est limitée par la définition de processus pré-programmés en nombre limité. On aura, dans tous les cas, recours à la programmation, dans des langages comme FORTRAN ou C, de sous programmes pour modéliser des règles ou des algorithmes de gestion et des fonctionnements particuliers.

V.3.4 Approche par objets

Les trois approches précédentes sont basées sur la notion d'événement. La modélisation d'un système de production, même s'il n'est pas complexe, ne peut raisonnablement pas se faire en juxtaposant simplement toutes les logiques de changements d'état du système. L'approche par objets consiste à modéliser le système par un ensemble d'objets qui dialoguent entre eux par

envoi de messages. Elle est basée essentiellement sur la notion de processus (certains simulateurs à objets sont basés sur la notion d'événements comme dans OASYS [BEL 90] et CADENCE). Cette approche permet de séparer très nettement les rôles respectifs du concepteur et de l'utilisateur. Le concepteur s'occupe de définir les différentes classes, tandis que l'utilisateur n'a plus qu'à créer des instances de ces classes pour son application. En effet l'étape de programmation qui suit normalement celle de modélisation n'a plus sa raison d'être. Le processus se trouve réduit et se résume à: modélisation, simulation, analyse des résultats.

V.4 Langages de Simulation

Pour simuler, on peut utiliser des langages évolués, des langages dédiés, des langages généraux de simulation ou des langages à objets.

L'utilisation d'un langage évolué (FORTRAN, PASCAL, C, etc.) présente l'avantage d'une grande souplesse. Nous programmons notre modèle à notre façon, avec des entrées/sorties interactives, etc. Mais les inconvénients de l'utilisation de tels langages dans des systèmes complexes font que l'emploi de cette approche est impossible: programmation nécessitant une analyse très fine, temps de programmation et surtout de validation importants et impossibilité de réutilisation d'un programme conçu pour un projet auparavant.

Les langages dédiés sont conçus pour un domaine d'application particulier; par exemple, l'usinage (MAP/1), le convoyage (SAME/AGVS), l'ordonnancement (PARSIFAL), les cellules flexibles (SIMUFLEX) [CAVAILLE et al. 87]. Ces langages permettent de décrire la structure d'un système de façon interactive sans avoir à connaître les langages de programmation. Les éléments de base d'un langage dédié sont très marqués sémantiquement (poste de montage ou d'usinage, convoyeur, navette, etc.). L'utilisation de tels langages n'est pas très répandue pour des raisons de non portabilité des modèles et des limites de la modélisation dues à la nature et au nombre des éléments de base contenus dans la bibliothèque d'un langage donné.

Les langages généraux de simulation ont un vocabulaire plus abstrait que les langages dédiés (files d'attente, ressources, stations, etc.), de façon à ne pas dépendre d'un domaine d'application. Ceci conduit à une grande souplesse de modélisation et de modification des modèles. Par contre, la phase d'apprentissage de tels langages peut être longue et la modélisation de règles de gestion complexes n'est modélisable qu'à travers l'utilisation des procédures écrites dans un langage évolué.

Les langages orientés-objets permettent d'unifier les données et les fonctions ou procédures en une seule entité: l'**objet**. Le premier langage orienté-objets est un langage de simulation:

SIMULA; mais la réussite de cette approche est due au langage Smalltalk pour ses qualités au point de vue du génie logiciel: convivialité, réutilisabilité, modularité, etc. ([BAILLY et al. 87] et [MASINI et al. 89]). La modularité de l'objet et sa spécialisation favorisent la flexibilité et la maintenabilité du modèle [AMAR 90]. Le polymorphisme et la liaison dynamique (résolution tardive) permettent de décrire les systèmes de production de façon évolutive et de l'exploiter avec des stéréotypes de conduite [BEL 90]. L'objet peut être spécialisé comme on le désire sans toucher aux autres objets grâce à la stabilité des interfaces (encapsulation). En outre, l'héritage et l'abstraction de données supportent les différents niveaux d'abstraction et facilitent la réutilisation des composants du logiciel [YE et al. 92a].

La plupart des langages orientés-objets sont séquentiels ([CAROMEL 93] et [MEYER 93]). Pourtant, les activités du système de production sont concurrentielles (parallèles). Un grand effort d'extension des langages à objets vers la programmation concurrente est en cours ([NELSON 91] et [WYAT et al. 92]). Ces langages facilitent l'implémentation des logiciels de simulation ([AKERBAEK 93], [NICOLAS 93] et [YE et al 94d]).

V.5 Etapes du Processus de Simulation

Plusieurs propositions existent dans la littérature. Un exemple de démarche, proposé par Duffau [DUFFAU et al. 84], se passe en quatre phases:

- analyse du problème et collecte des données;
- modélisation, programmation, vérification, validation;
- définition des expériences, exécution du modèle;
- exploitation des résultats.

L'application de ces étapes ne se déroule pas selon un processus strictement séquentiel. La démarche est au contraire de caractère très itératif.

Balci [BALCI 90] a proposé une autre démarche en dix étapes qui nous semble plus complète et plus naturelle (figure 2-18):

- formulation du problème;
- recherche de solutions techniques;
- description du système;
- formulation du modèle;
- représentation du modèle;
- programmation;

- élaboration d'expériences;
- expérimentation;
- redéfinition;
- présentation des résultats.

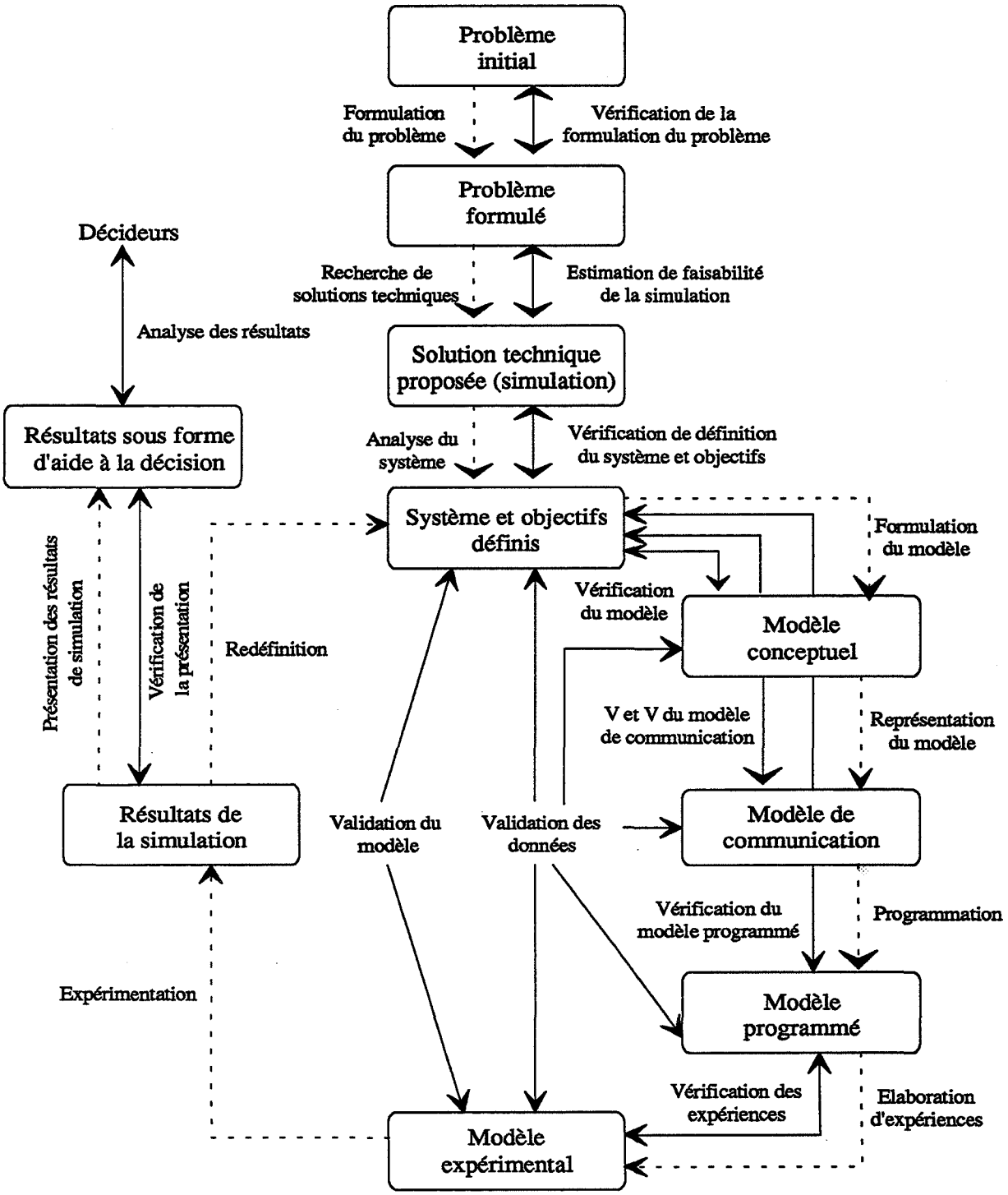


Figure 2-18 Etapes d'un Processus de Simulation

A chacune de ces phases est associée une procédure de vérification et/ou de validation. La vérification consiste à examiner si le modèle de simulation est bien construit. La validation consiste à tester si on a construit le bon modèle de simulation.

L'application des étapes du processus de simulation demande la résolution de différents problèmes pour lesquels on ne dispose pas ou peu d'outils d'aide. Les principaux problèmes se situent au niveau de la modélisation, la validation statistique et l'interprétation des résultats. Dans ce mémoire, nous mettrons en évidence l'étape de modélisation et simulation en utilisant l'approche à objets et nous proposerons les étapes de modélisation suivantes [YE et al. 94c].

Etape 1: Identification des classes d'objets et de leurs services en utilisant des méthodes traditionnelles d'analyse et de conception par objets.
Etape 2: Séparation des objets actifs et passifs. Les objets actifs sont des entités dont l'activité est autonome.
Etape 3: Définition de diagramme de transition d'états et de diagramme de communication de processus pour les objets actifs.
Etape 4: Définition d'activité (comportement) d'objets actifs et adaptation des contraintes de pilotage (en intégrant des règles de conduite et de gestion).
Etape 5: Raffinement des attributs et des services d'objets de l'application et construction de modèles de simulation.

V.6 Conclusion sur la Simulation

Les modèles de simulation sont capables de décrire le système avec le degré de détail et de précision nécessaire qui convient à la résolution du problème posé. Cette description inclut la partie physique de l'atelier mais aussi certains aspects du système de gestion de production.

Les logiciels de simulation tendent à se multiplier aujourd'hui. Ils ont pour objectif d'être plus proches des besoins des utilisateurs, plus faciles à utiliser et plus puissants. Les recherches menées actuellement s'orientent dans de nombreuses directions. Notons en particulier que plusieurs études sont conduites actuellement pour appliquer la technologie à objets, la programmation concurrente et distribuée dans la construction des modèles de simulation. L'utilisation des approches à objets devrait permettre une avancée importante dans le domaine de la simulation, en permettant la mise au point d'un outil de modélisation à la fois modulare, flexible et convivial [BEL 90]. La programmation concurrente permettra d'implémenter facilement la coopération (communication et synchronisation) entre les objets dans un modèle de simulation orientée-objets.

VI Conclusion

Dans ce chapitre, nous avons survolé les principales méthodes d'analyse et de conception des systèmes de production. On peut les regrouper en termes des méthodologies informatiques en trois catégories: méthodes basées sur la décomposition fonctionnelle, méthodes basées sur les "data-flow diagrams" et méthodes basées sur l'approche à objets [COAD et al. 90].

La *décomposition fonctionnelle* conduit les concepteurs à commencer par la description du système la plus abstraite du point de vue de sa fonction puis de raffiner cette vue par des étapes successives (méthodes SADT, GRAI et CIMOSA). Chaque sous-système est décomposé dans chaque étape en un petit nombre de sous-systèmes plus simples que leurs ancêtres, jusqu'à ce que tous les éléments déduits soient de nouveau d'un niveau d'abstraction suffisamment bas afin de les implémenter directement. C'est une *démarche d'analyse et de conception descendante*, c'est-à-dire que l'on part de la description (fonctionnelle) la plus générale du système et que l'on incorpore les détails au fur et à mesure de l'avancement de l'analyse et de la conception.

Cette démarche est très difficile à construire et très volatile. Du point de vue du génie logiciel, elle est fiable, efficace, etc., mais non réutilisable, non évolutive, non maintenable. Du point de vue de l'approche à objets, cette méthode:

- est généralement inadéquate pour les problèmes de simultanéité (concurrency nature);
- n'utilise pas le principe de type abstrait et d'encapsulation;
- est souvent difficile à modifier (non-évolutive) suivant le changement de la spécification correspondante;
- ne permet pas le partage du code (héritage);
- manque la flexibilité;
- etc.

L'approche de *data flow diagram*, souvent nommée une méthode *d'analyse et de conception structurée* (méthodes SA/SD, MERISE), est une autre méthode de transformation d'un problème réel en une représentation technique. Cette transformation exige des analystes (et plus significativement des clients), un suivi des flux de données chaque fois qu'ils observent le monde réel et traduisent ce flux en sous-séquence analyse et spécification. Bien que "follow the flow of data" ne soit pas l'une des méthodes essentielles que les concepteurs (analystes) utilisent pour résoudre la complexité du problème, ce flux décrit plus en détail les processus-étapes et ils sont très utiles pour identifier les méthodes et les attributs d'objets dans l'approche à objets. La problématique de cette approche est:

- la difficulté de sélectionner les attributs appropriés et leur nombre;

- la taille de son "data dictionary";
- les interfaces volatiles de données;
- la difficulté à transformer le data flow diagram en programme;
- etc.

L'approche à objets, issue du génie logiciel, permet de décrire concrètement un système de production. Les concepts et les processus de mise en oeuvre de cette approche seront discutés dans les deux chapitres suivants. Nous présentons ici seulement pourquoi nous avons adopté l'approche par objet pour la simulation [YE 93].

- Exigence du génie logiciel: abstraction de données, modularité, encapsulation, réutilisabilité, compréhensibilité, ... ;

- Description concrète du système à simuler: Possibilité de **one-to-one-mapping** entre les objets à modéliser dans l'industrie manufacturière et leurs abstractions dans un modèle de simulation;

- Structuration du système de conduite [BEL 90]: Le développement des classes d'objets du système de production manufacturière implique que ces objets modélisés maintiennent une quantité importante de données de management de production et de règles à choisir (méthodes) dans les systèmes comme M.R.P., O.P.T., J.A.T., etc.

- Description évolutive de l'atelier: Si les classes d'objets essentielles du domaine sont construites, la tâche de construction du modèle réel est allégée, c'est-à-dire que le temps de construction ou composition (model **setup time)** du modèle est réduit; puis, en fonction de l'évolution de la conception, les objets de base sont spécialisés ou remplacés progressivement par des objets plus détaillés, introduisant éventuellement de nouvelles données à définir et de nouvelles méthodes (ou de méthodes surchargées).

- Stockage de stéréotypes de composants et de systèmes de conduite [BEL 90]: La disponibilité de ces classes du domaine permet de modéliser rapidement et précisément un système dans différents niveaux d'abstraction (atelier, poste de travail, machine).

- Dès que ces classes d'objets hiérarchisées sont développées, les règles ou méthodes de management appliquées sur ces objets peuvent être encapsulées comme des méthodes d'objets. Par exemple, les concepts de J.A.T. telles que des règles "tirer" ou "pousser" le flux, réduire le temps de préparation, minimiser les en-cours, etc. peuvent être codés comme des méthodes dans les objets machines et stations.

- Modélisation facilitée des systèmes de management de production complexes grâce aux concepts d'héritage, de méthodes surchargées, de liaison dynamique, etc.

Chapitre III Analyse des Systèmes de Production par l'Approche Objet

I Introduction

L'*analyse*, première étape du développement d'une application, consiste à définir et à modéliser les problèmes d'un système [MCGREGOR 92] dans le but de fournir une description du comportement externe observable. La méthode d'analyse procédurale ou fonctionnelle produit un cahier des charges qui spécifie les fonctionnalités du système désirées par les clients ou par les utilisateurs.

L'*analyse par objet* est une méthode utilisée pour identifier les entités significatives réelles ou conceptuelles d'un système, et pour comprendre et expliquer comment ces entités inter-réagissent afin de réaliser les objectifs visés par les utilisateurs [SHLAER et al. 92]. Le résultat d'analyse est un document qui décrit une caractérisation essentielle de l'espace du problème (un ensemble d'entités réelles ou conceptuelles) et leurs fonctionnalités.

Le processus d'analyse est subdivisé en deux étapes: l'*analyse du domaine* et l'*analyse de l'application*. L'analyse du domaine établit le contexte principal dans lequel ce système sera construit. L'analyse de l'application se concentre sur le domaine d'intérêt afin de construire un cahier des charges pour une application particulière. L'analyse par objet se distingue de la méthode procédurale par deux aspects: *le centre d'intérêt* et *le point de vue*.

L'analyse par objet fournit une possibilité de modéliser plus profondément et plus naturellement l'espace de problèmes en considérant une plus grande partie de son espace au lieu d'une partie particulière le concernant. Les concepts identifiés par l'analyse par objet seront plus stables et plus abstraits que ceux développés par l'analyse traditionnelle. Ces concepts sont des composants de base pour construire des modèles d'application plus flexibles et plus extensibles. Les modèles réalisés en utilisant ces abstractions s'accommodent plus facilement au changement de spécification (d'une application particulière); et ils facilitent la compréhension, par les utilisateurs, des problèmes à résoudre et de leurs solutions potentielles.

Le document produit en utilisant l'approche d'analyse par objet est très différent de celui de l'analyse procédurale. Les documents traditionnels sont orientés-fonctionnalité: un système est vu comme un ensemble de services qu'il peut fournir. Les documents "à objets" s'appuient sur

la description des entités et leurs relations avec la problématique du domaine: un système est considéré comme un ensemble d'entités qui inter-réagissent.

Avant de construire un système informatique, les analystes doivent en général traiter un grand nombre de différents sujets ou domaines. Chaque domaine peut être considéré comme un monde séparé qui possède ses propres entités conceptuelles ou objets, et qui peut exister plus ou moins indépendamment des autres. Les interactions entre ces domaines se réalisent par différents événements. Dans un système de GPAO, par exemple, le domaine de production concerne les produits, les machines, les transporteurs, les opérateurs, etc., tandis que le domaine de l'interface utilisateur est constitué de fenêtres, de boutons de contrôle, d'icônes et d'affichages textuels ou graphiques, etc.

L'ensemble des domaines du système est représenté par un diagramme de domaines (exemple en gestion de production assistée par ordinateur dans la figure 3-1). Certains domaines sont assez petits pour que l'on puisse les analyser intégralement, tandis que d'autres contiennent trop d'objets (éléments) et, doivent être divisés en différents sous-systèmes analysables.

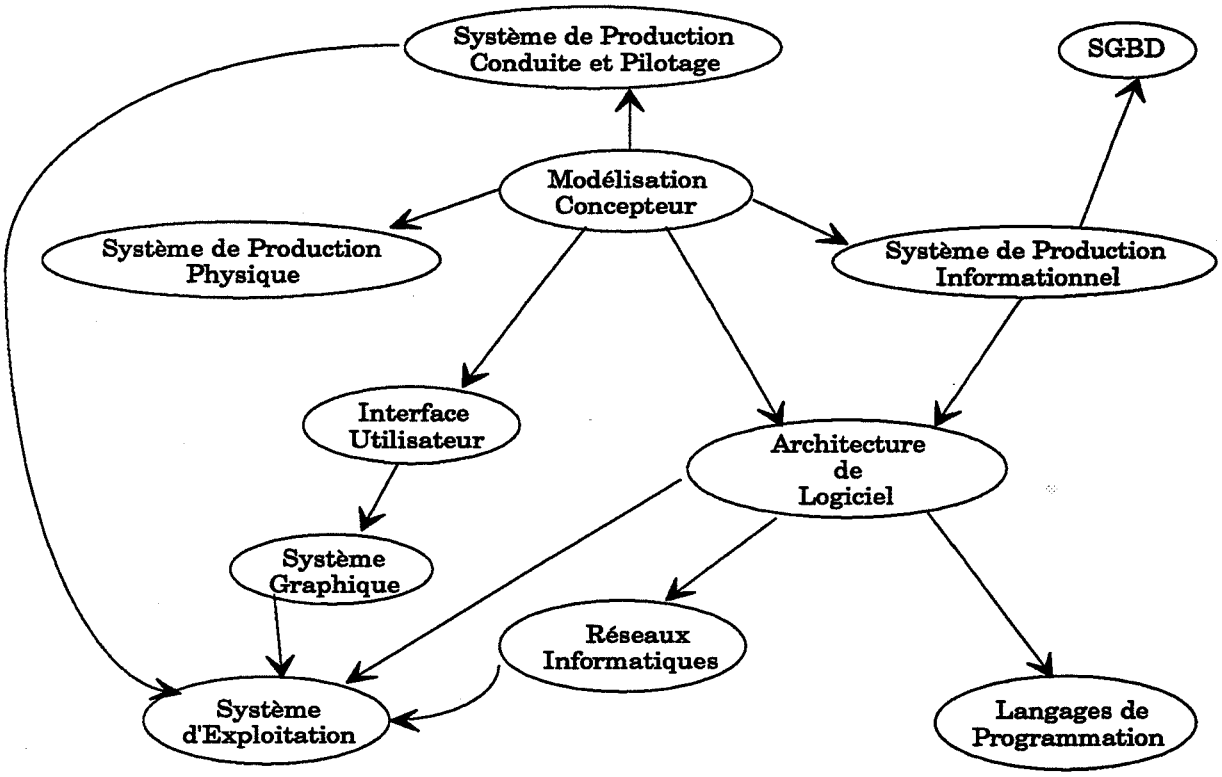


Figure 3-1 Diagramme des Domaines de GPAO

Une fois le système décomposé en domaines et en sous-systèmes, l'analyse du domaine peut être entreprise.

II Analyse du Domaine.

L'analyse du domaine est l'activité d'identification des objets et des services d'un ensemble de systèmes similaires pour un domaine d'application. C'est un processus dans lequel les informations utilisées dans le développement de logiciels sont identifiées, recueillies et organisées pour rendre ces informations réutilisables quand on met à jour ou quand on crée de nouveaux systèmes similaires [PRIETO 90].

II.1 Définition du Domaine

Un domaine est un monde réel séparé et hypothétique, ou un monde abstrait, qui constitue un ensemble d'objets distincts qui se conduisent conformément aux règles et aux caractéristiques du domaine. En général, un domaine est "une sphère d'activités ou d'intérêts: *champs*" [SHLAER et al. 92]. En termes de génie logiciel, il peut être interprété comme un champ d'application dans lequel les logiciels sont développés.

Un domaine peut être simple comme un algorithme de calcul ou compliqué comme un système de GPAO qui est un ensemble de sous-domaines imbriqués ou semi-hiérarchisés (figure 3-1). Les domaines sont classés en quatre types selon leur rôle dans un système informatisé.

1) **Les domaines d'application** sont le sujet du discours issu de la spécification des clients (utilisateurs) du système: quelles sont les fonctionnalités dont les clients ont besoin. Pour un projet donné, il n'y a normalement qu'un domaine d'application.

2) **Les domaines de service** fournissent les mécanismes généraux et les fonctions utilitaires afin de supporter les domaines d'application. Notons que, au moment de l'analyse, chaque domaine de service fait certaines hypothèses concernant les autres domaines dans le système. Ils seront complétés et modifiés plus tard dans la phase de conception et d'implémentation. La figure suivante (figure 3-2) [CUGY 83] montre une organisation fonctionnelle des domaines de services d'une entreprise. Chaque domaine est défini par son objectif et les moyens mis à sa disposition pour l'atteindre.

- Le domaine *vente* assure la liaison entre l'entreprise et ses clients. Le processus de vente ne correspond plus à faire écouler sur le marché des produits déjà conçus et fabriqués mais à analyser les besoins des clients (étude de marché) à partir desquels sera faite la conception d'un nouveau produit et sa satisfaction;

- Le domaine *technique* comprend la conception des produits et l'établissement des phases d'industrialisation des produits (nomenclatures, gammes, etc.);

- Le domaine *production* met en oeuvre les moyens de production pour obtenir les produits finis destinés à la vente;

- Le domaine *approvisionnement* assure l'alimentation des ateliers de production en matières premières et ressources;

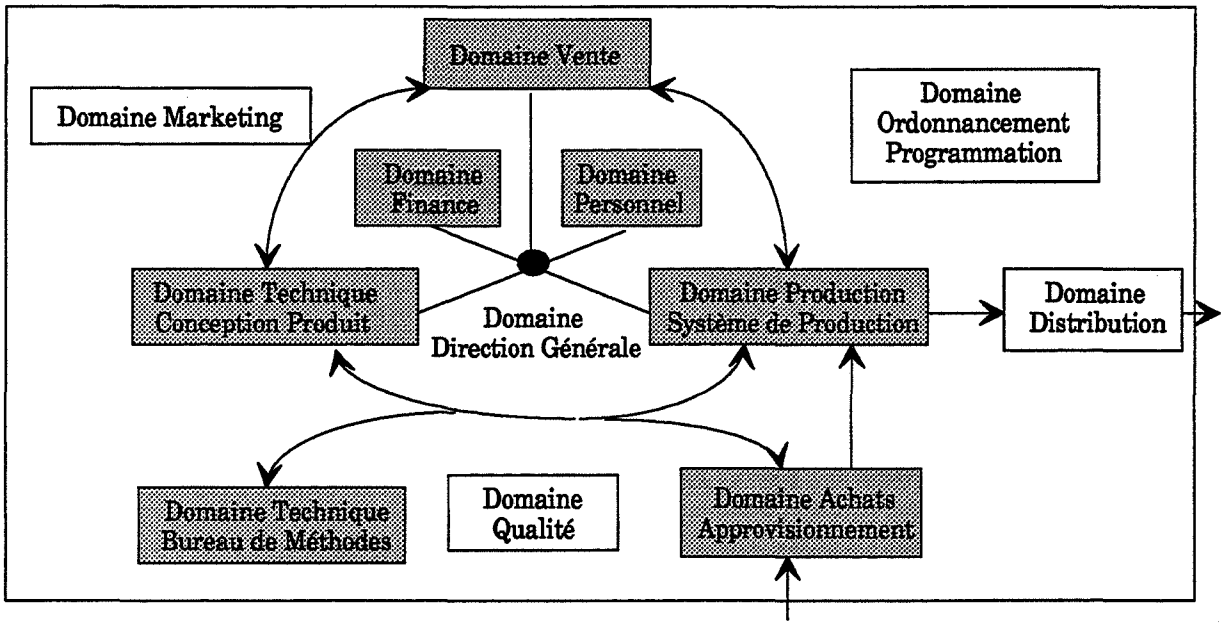


Figure 3-2 Fonctionnalités Principales d'une Entreprise

- Le domaine *personnel* gère les ressources humaines de l'entreprise;

- Le domaine *finance* et *comptabilité* a pour but de se procurer et de gérer les capitaux nécessaires au fonctionnement de l'entreprise ou à son expansion et d'enregistrer les mouvements de fonds.

3) **Les domaines d'architecture** fournissent les mécanismes et les structures généraux pour gérer les données et contrôler intégralement le déroulement du système. Les objets dans les domaines d'architecture comprennent les abstractions de structures de données élémentaires et d'unités de code de leur gestion ou de leur contrôle. La figure suivante (figure 3-3) montre une architecture de simulation par processus avec la notion de *thread* supporté par le langage de programmation C++. Cette architecture a plusieurs objectifs.

Le premier est d'imposer une *uniformité* au niveau de la construction du logiciel par des stratégies qui répondent aux questions suivantes.

* Comment les données sont-elles organisées et distribuées?
* Comment le flux de contrôle est-il ordonné, structuré et enchaîné?
* Comment l'application et le code de service sont-ils structurés?
* Quelles intercommunications sont autorisées entre quelles unités de code?
* etc.

Leur intention est de limiter la complexité du système à construire en fournissant des structures normalisées et des voies communes pour des développements des systèmes similaires.

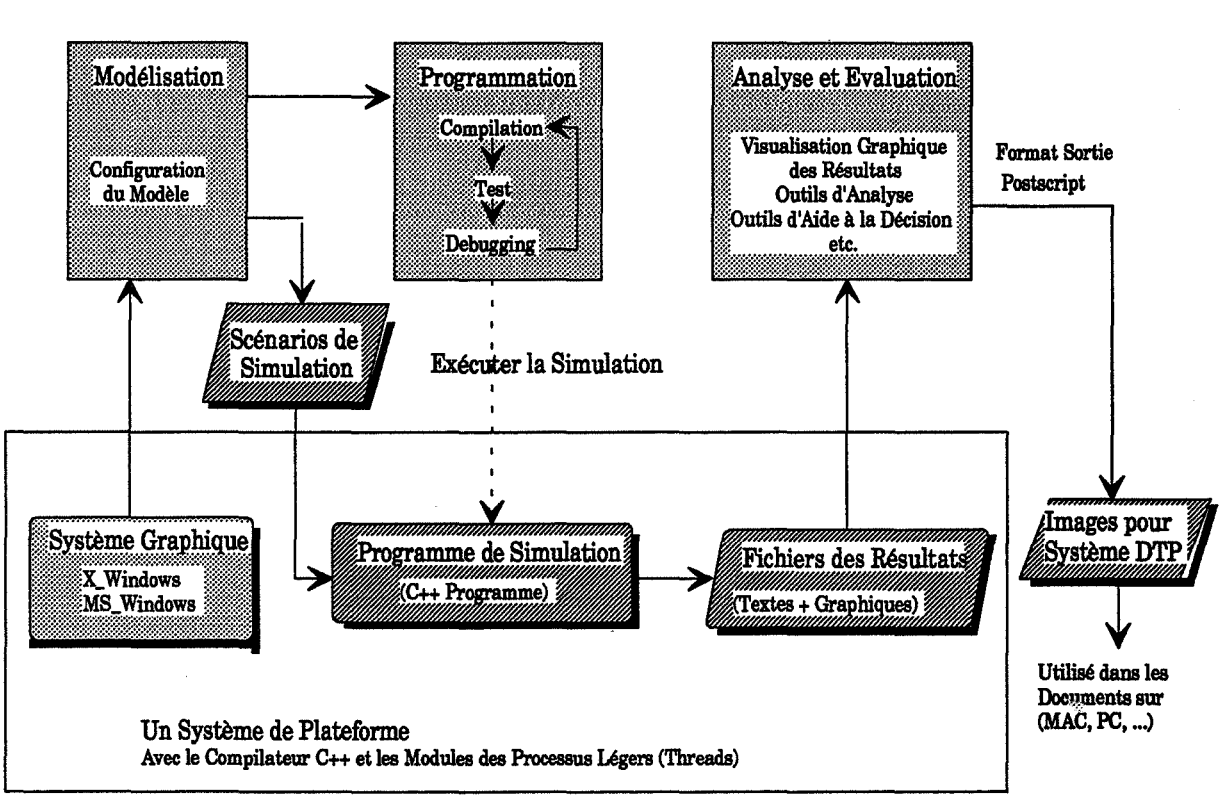


Figure 3-3 Architecture de Simulation Concurrente avec des Threads

Le deuxième objectif est de définir les différentes *activités du système* au niveau de l'architecture. La plupart des systèmes exigent des codes signifiants pour gérer l'initialisation et la fermeture du système, pour maintenir la coordination des différentes activités et pour transférer les données entre les différents sous-domaines. Ces activités influent sur tout système; il ne faut pas les mélanger avec les domaines d'applications et les domaines de

services. Le domaine d'architecture peut donc donner une structure complète pour définir et organiser ces activités communes de façon plus cohérente.

Le troisième objectif est de définir une *plate-forme* (une interface d'architecture avec plusieurs systèmes d'exploitation) pour le problème de *portabilité*. Le schéma de la figure 3-3 montre bien que cette architecture s'adapte à tous les systèmes qui possèdent un compilateur C++ et un module de manipulation des processus "légers" (threads).

Enfin, nous pouvons construire des composants ou des modules standardisés et validés ou bien des *sous-systèmes embarqués* si nous avons une architecture commune pour le but d'optimisation de performances (abstraction, réutilisabilité, modularité, etc.).

4) **Les domaines d'implémentation** incluent les langages de programmation, les réseaux informatiques, les systèmes d'exploitation et les librairies de classes communes. Ces domaines fournissent les entités conceptuelles nécessaires à l'implémentation du système. Cet aspect concernant l'implémentation de la simulation concurrente des systèmes de production par l'approche à objets sera détaillé dans le chapitre V.

II.2 Processus de l'Analyse du Domaine

Un système complexe est composé de plusieurs domaines imbriqués. Chaque domaine est limité par une frontière dans laquelle sont définis les objets, les opérations et les relations internes et externes. Cette frontière délimite la capacité opérationnelle de chaque domaine.

L'analyse du domaine est un processus dans lequel les informations utilisées dans le développement de logiciels sont identifiées, capturées et organisées dans le but de rendre ces informations réutilisables dans le futur. Plus précisément, l'analyse du domaine essaie de développer et d'évaluer une *infrastructure* d'information (comme dans le modèle CIMOSA) afin de fournir un standard et de permettre la réutilisation des informations. Les composants de cette infrastructure comprennent les modèles du domaine, les standards du développement et les librairies des composants réutilisables.

Malheureusement, l'analyse du domaine est conduite d'une manière *ad hoc*. Le processus d'abstraction est habituellement considéré comme une activité intelligente de l'être humain et il est très associé avec les "expériences". Prieto-Diaz [PRIETO 90] a donné un schéma SADT (figure 3-4) qui montre les entrées, les sorties, les contrôles et les mécanismes pour décrire le contexte de l'analyse du domaine.

Les informations sont collectées à partir des systèmes existants comme les programmes sources, la documentation, le manuel utilisateur, etc., en même temps que la connaissance du domaine et le cahier des charges du système actuel et futur. Les techniques de l'intelligence artificielle comme l'acquisition ou la représentation des connaissances jouent un rôle important. Les experts et les analystes du domaine extraient les informations et les connaissances (figure 3-5) en s'appuyant sur les systèmes existants, les méthodes d'analyse et d'autres informations introduites. Avec l'aide des ingénieurs du domaine, ces connaissances et abstractions sont organisées et encapsulées sous la forme des modèles du domaine, des standards et des collections de composants réutilisables.

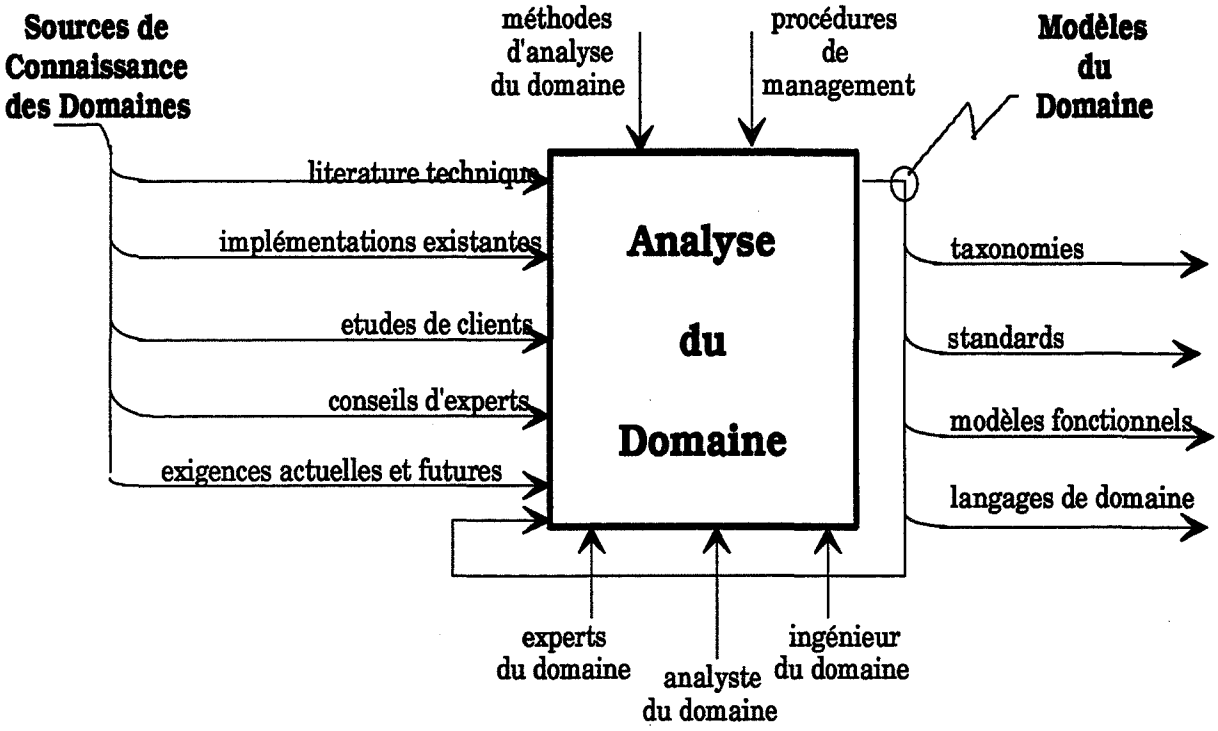


Figure 3-4 Contexte de l'Analyse du Domaine

C'est un processus de raffinement itératif et incrémental. Les modèles du domaine, sous différentes formes, supportent différentes étapes du développement d'un logiciel. Les ressources réutilisables sont sélectionnées et intégrées dans les nouveaux systèmes. Les données réutilisables sont ensuite collectionnées et retournées à la phase d'analyse du domaine afin de raffiner et d'enrichir les modèles du domaine et de mettre à jour la librairie des ressources et des données.

Trois intervenants autres que les experts du domaine, les analystes du domaine et les ingénieurs du domaine jouent un rôle important dans ce processus. Ce sont le gestionnaire de ressources ou de moyens, le responsable maintenance et le bibliothécaire. La tâche du bibliothécaire est de

rendre les ressources disponibles et faciles d'accès pour les futurs utilisateurs. Le gestionnaire de ressources règle la qualité de ces ressources et les normalise. Le responsable maintenance maintient la collection des données concernant l'analyse du domaine et coordonne la réutilisabilité globale du système.

En bref, l'objectif de l'analyse du domaine est de créer un système général ou un ensemble de systèmes intégrés tout en fournissant une infrastructure en permettant que l'on capture et représente les objets des applications potentielles.

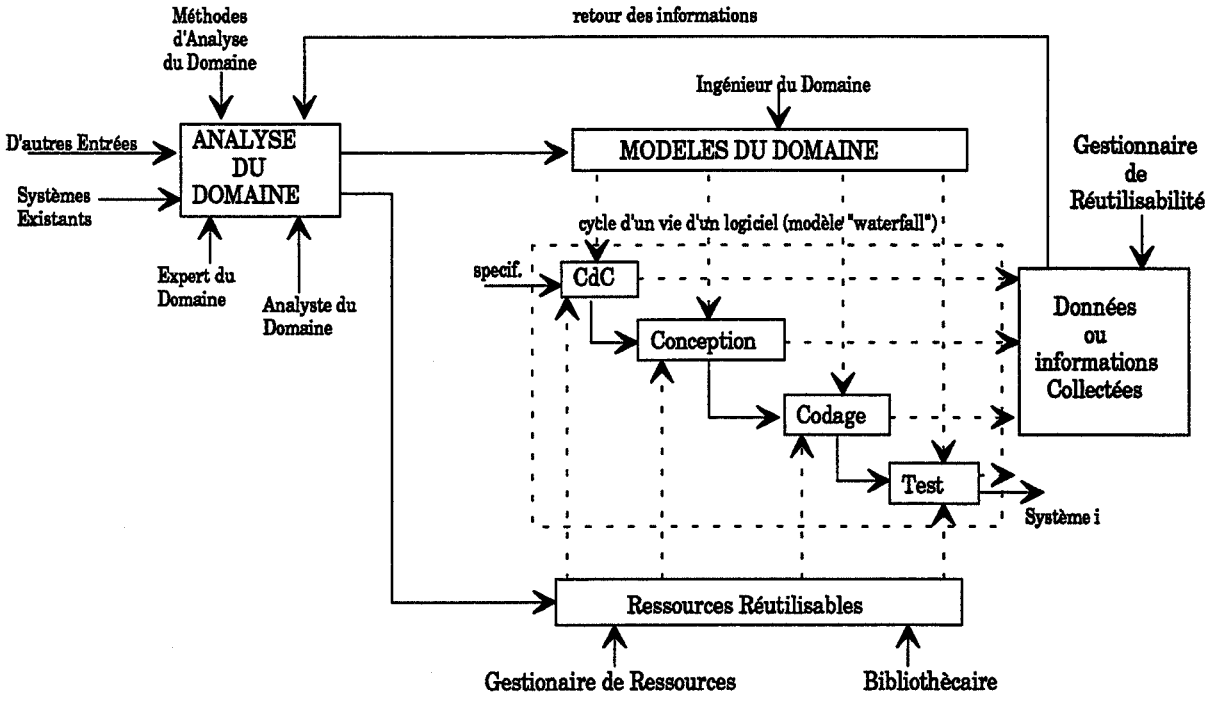


Figure 3-5 Infrastructure de Réutilisabilité de l'Analyse du Domaine dans le Cycle de Vie d'un Logiciel

II.3 Résultat de l'Analyse du Domaine

L'analyse du domaine est considérée comme un challenge stratégique ainsi qu'une technique dans laquelle les analystes essaient d'aboutir à un consensus sur le vocabulaire et la théorie d'un domaine [TRACZ 92]. Pour cela, certains moyens de représentation tels que: 1) langages naturels, 2) schéma entité-association, 3) objets, 4) thesaurus/taxonomie, 5) logique des prédicats, et 6) réseaux sémantiques ou d'autres techniques de représentation issues de l'IA, sont utilisées pour construire une abstraction des objets du domaine. Krueger [KRUEGER 92] divise les approches d'abstraction en huit catégories: 1) langages de haut-niveau, 2) réutilisation directe des concepts et des codes par expérience (design and code scavenging), 3) composants des codes, 4) schémas des algorithmes et des structures de données (software

schemas), 5) générateurs d'applications, 6) langages naturels ou langages de spécification, 7) systèmes transformationnels et 8) architectures de logiciels.

Il est difficile de comparer dans l'absolu les différentes approches existantes. Elles se distinguent en premier lieu par leur *champ d'application*; c'est-à-dire que certaines approches sont plus spécialisées, et donc plus à même de prendre en compte certains composants du système que d'autres. Si la conception et la programmation par objet concernent plutôt les composants codes, l'analyse par objet se concentre sur les composants (classes d'objets) du domaine et leurs interactions ou plutôt leurs relations. En gestion de production, Il est classique de décomposer les systèmes de production en trois composants principaux.

1) Le **système physique** (S.P.), appelé aussi système opérant ou système technologique, agit sur les produits en effectuant des transformations, des contrôles, des stockages et des opérations de manutention. Les systèmes physiques sont organisés de différentes façons, par exemple en lignes flexibles, chaîne de transfert, îlot de production, etc.

2) Le **système de décision** (SD), appelé aussi système de conduite ou système de pilotage, a pour but de modifier, par ses décisions, l'évolution du système physique. Il décide en fonction: du comportement du système physique, de l'état de l'environnement et des objectifs qui lui ont été assignés (par exemple, fabriquer une quantité donnée d'un produit dans un délai prévu suivant une qualité demandée). Les décisions peuvent être humaines, assistées par ordinateur (système d'aide à la décision, systèmes experts) ou automatisées (système intégré de production). Parmi les décisions dont ce système a la charge, on distingue de façon classique les décisions de planification, d'ordonnancement, et de lancement.

3) Le **système d'information** (SI), assure essentiellement la collecte, la transmission, le traitement et la mémorisation des informations venues de l'environnement du système de production, mais également du système lui-même. Il sert de liaison entre le système de décision et le système physique.

La figure 3-6 illustre les liaisons entre ces trois composants. Nous nous intéressons particulièrement à la modélisation du système physique et aux informations à collecter pour que l'on puisse prendre des décisions. L'intégration du système de décision sera considérée dans la modélisation des processus des moyens de production dans le chapitre IV.

Un système physique est composé de trois sortes de populations: les produits à transformer, les moyens de production (transformation et manutention) et les ressources qui conditionnent le flux de produits sur les moyens de production (figure 3-7). Ces trois types de population, que

l'on retrouve invariablement selon les répartitions les plus diverses dans toutes les unités de production, sont les éléments de base de tout système de production. Dans ce mémoire, nous essayons de les modéliser au niveau opérationnel dans le but de construire un modèle de simulation des systèmes de production. On s'intéresse plutôt à l'axe processus de la production ([JULLULIEN 92], [MATHON 92]). Les outils et les opérateurs sont considérés comme des ressources afin de mieux définir les processus technologiques qui constituent le coeur de la production.

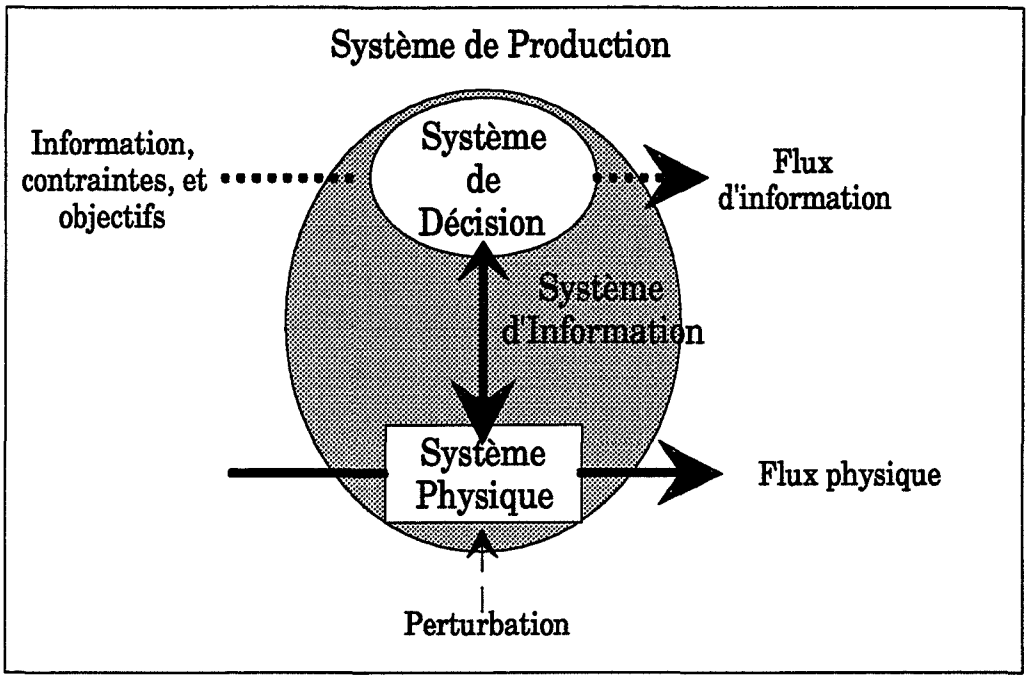


Figure 3-6 Représentation Simplifiée du Système de Production

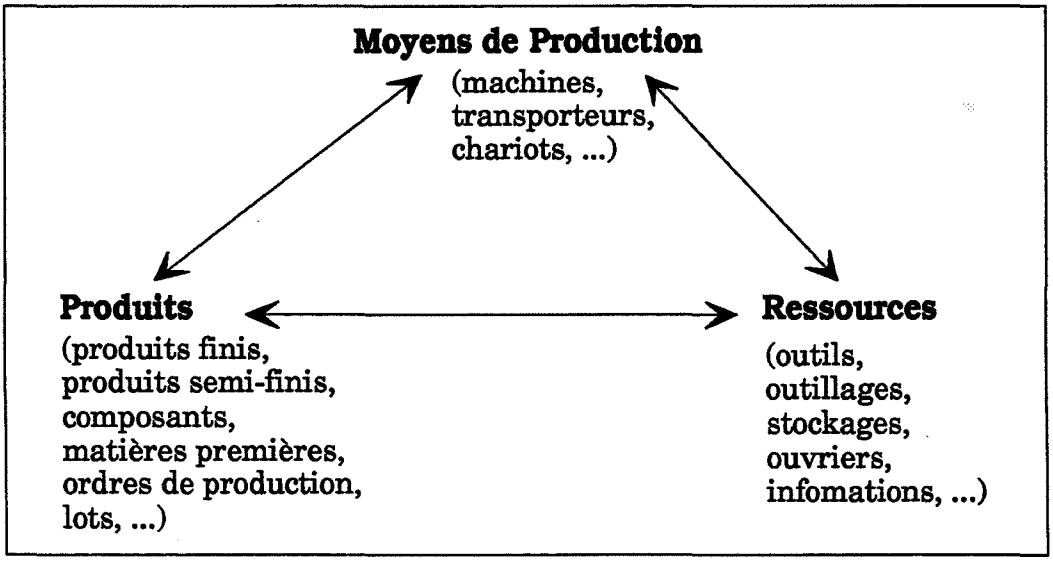


Figure 3-7 Trois Populations du Système Physique (Point de Vue de la Simulation)

II.4 Identification des Classes d'Objets du Domaine

Dans cette phase d'analyse, les objets sont identifiés du point de vue des utilisateurs. Il n'est pas évident de construire une abstraction des objets similaires bien que l'on puisse identifier les individus facilement dans le monde réel. Par exemple, dans un atelier, on voit différents moyens de production tels que des tours, des fraiseuses, des convoyeurs, etc.; mais comment peut-on les abstraire en des classes qui possèdent des caractéristiques communes. Une solution possible par exemple, en considérant l'espace du problème et les différentes exigences informatiques pour la simulation, on peut abstraire ces moyens de production comme un processus, puisque ces moyens absorbent des articles en entrées (pièce ou composant, matière première, lot, etc.), et les restituent en sortie après un délai dépendant de leurs comportements et des articles traités. Tsichritzis [TSICHRITZIS 1988] propose cinq catégories d'objets à identifier dans les systèmes à objets: 1) objets physiques, 2) objets "rôles", 3) objets "incidents", 4) objets "interactions" et 5) objets "spécifications".

1) Les **objets physiques** sont des entités réelles qui existent dans le système. Nous pouvons les apercevoir facilement, et leurs attributs peuvent être identifiés et déterminés à travers leur comportement et les différents points de vue des utilisateurs. Par exemple, l'activité de la production, d'une manière générale, met en jeu de façon interactive des individus appartenant à trois populations: la population des produits, des moyens de production et des opérateurs de production. Elles forment, en quelque sorte, un espace dans lequel se déroulent les opérations de production.

Les attributs sont de différentes natures: *structurels*, *conjoncturels*, *événementiels*, *instantanés*, *historiques* et *statistiques* [YE et al. 93]. Pendant la phase d'analyse, on peut identifier les attributs structurels et conjoncturels des objets. Les attributs structurels, peu évolutifs, caractérisent les aspects fondamentaux des objets. Certains d'entre eux décrivent les liens logiques avec d'autres objets et leur spécification fine dépend essentiellement de la nature des applications et des conditions d'exploitation. Les attributs conjoncturels expriment le contexte dans lequel doit se dérouler la production. Ils sont, dans une large mesure, non modifiables dans le cadre de la production elle-même et représentent plutôt les contraintes extérieures. La figure suivante (figure 3-8) montre des exemples d'attributs essentiels d'un opérateur. L'attribut *qualification* est une donnée fondamentale du système de production pour représenter le niveau de compétence d'un opérateur. L'attribut *niveau de rémunération* sert à reconstituer les coûts opérationnels de production. L'attribut *moyens affectés* sera utilisé par des règles d'allocation des ressources. Le *calendrier des présences* exprime la disponibilité d'un opérateur

en fonction du temps; il est complété normalement par un tableau d'affectation selon les secteurs d'organisation de production qu'on va ajouter à la phase d'analyse de l'application.

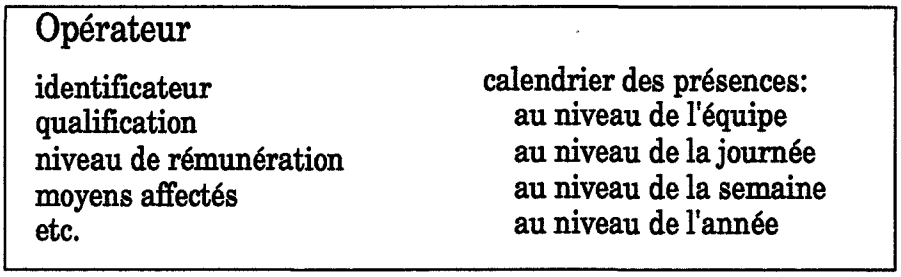


Figure 3-8 Attributs Structurels et Conjoncturels d'un Opérateur

2) Les **objets "rôles"** modélisent les contraintes auxquelles sont soumises les objets ou les règles d'utilisations d'objets. Ces objets sont en général des attributs des autres objets physiques. Dans la production, par exemple, les objets tels que le programme de fabrication,, la nomenclature de produit et la gamme d'usinage de pièce, sont des contraintes ou règles opératoires. Les objets comme la prévision de demandes, la planification de production, les règles de priorité et de pilotage sont des fonctions que le système de gestion de production doit accomplir. Une autre utilisation des objets rôles est d'exploiter les relations au sein des méthodes d'objets. Par exemple, le concept de "lot" joue le rôle des entités de flux dans la simulation des flux de production; et les fonctions telles que des règles de priorité seront choisies dynamiquement par la machine selon l'état de la file d'attente, l'état de la machine et l'objectif de la production.

3) Les **objets "incidents"** représentent les apparitions des événements aléatoires. Un objet opération de fabrication, par exemple, contient des informations sur le moyen de production utilisé, le temps de préparation, le temps opératoire, la date de début et de fin, les outillages utilisés, etc. Un objet incident d'une opération de "panne" est produit quand il y a une casse d'outil, un blocage d'un mouvement d'outillage, une panne de machine de toute nature: mécanique, électrique, électromécanique, pneumatique, informatique, etc.

4) Les **objets "interactions"** représentent des liens ou des relations entre les différents objets. Des liens persistent systématiquement entre les différents individus du monde réel. Une relation est un lien stable entre deux individus différents, qui permet à tout moment, connaissant l'un d'entre eux, de connaître d'un autre. Il y a deux natures de relations: conditionnelle et inconditionnelle [SHLAER et al. 92]. La relation *inconditionnelle* exige que chaque partenaire d'une association participe à la relation. Dans la relation *conditionnelle*, un ou l'autre des partenaires d'une association n'est pas obligé de participer à la relation. Une autre caractéristique d'une relation est sa *cardinalité* qui représente le nombre d'instances d'un côté

qui sont liées avec une instance d'un autre côté. En incluant la cardinalité et la forme conditionnelle et inconditionnelle, dix types différents de relations existent entre deux classes d'objets partenaires (figure 3-9).

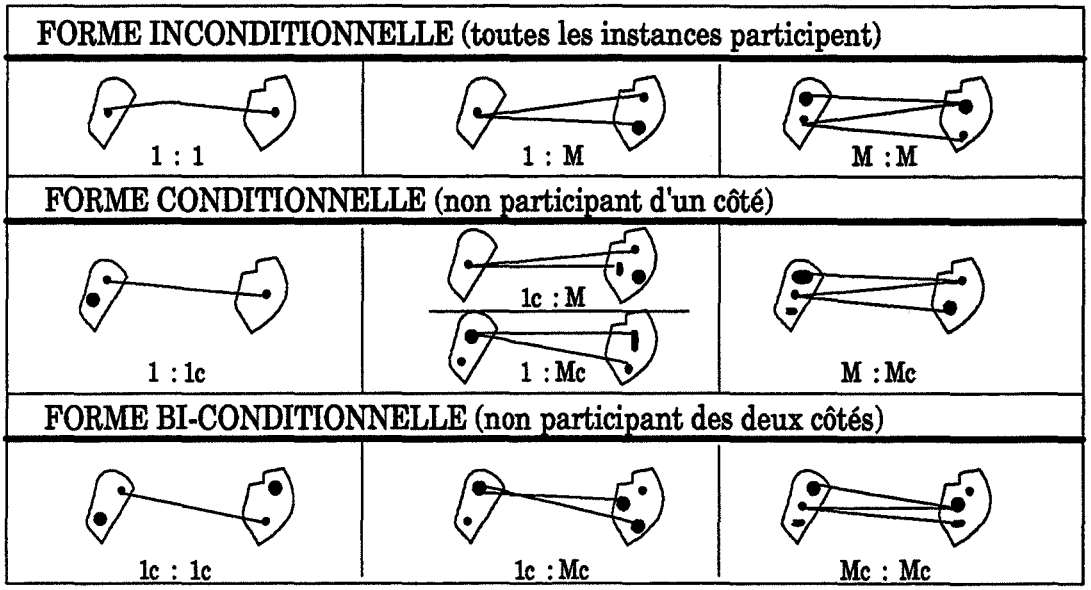


Figure 3-9 Dix Formes d'une Relation entre Deux Classes Partenaires

Le but d'une relation est de permettre la mise en évidence des instances d'une classe qui sont associées avec celles d'une autre classe. Cela est fait en plaçant les attributs de type référence (un pointeur en terme informatique) dans les classes partenaires. Dans le cas de la cardinalité un-à-un, l'attribut référence peut être ajouté dans l'un des deux classes. On place un attribut référence dans le côté plusieurs d'une relation si la cardinalité est un-à-plusieurs. Afin de formaliser la relation plusieurs-à-plusieurs, nous devons créer un objet "interaction" ou un objet "association" qui contient les références de l'identificateur de chaque classe participante. Cet objet possède son propre identificateur, ses attributs et ses méthodes. Une commande de produit, par exemple, contient les attributs: identificateur, référence de clients, référence de produits, quantité commandée, date de commande, délai de livraison, etc. Les méthodes pour définir son comportement sont illustrées dans la figure 3-10.

5) Les **objets "spécifications"** représentent des points de vue d'une partie sélectionnée d'une structure ou des supports qui combinent des entités simples en entités plus complexes. L'objet spécification fait habituellement de référence aux autres objets. Un tableau d'affectation des opérateurs aux moyens de production, un catalogue des articles stockés dans le magasin, une file d'attente amont et aval d'une machine, différents tableaux de bord, etc., appartient à ce genre des spécifications. Les points de vue d'un objet correspondent à la notion de version d'objet: un produit en stock, en-cours de fabrication, en ordonnancement, par exemple,

possède la même structure de base mais leurs interprétations contextuelles ne sont pas forcement les mêmes.

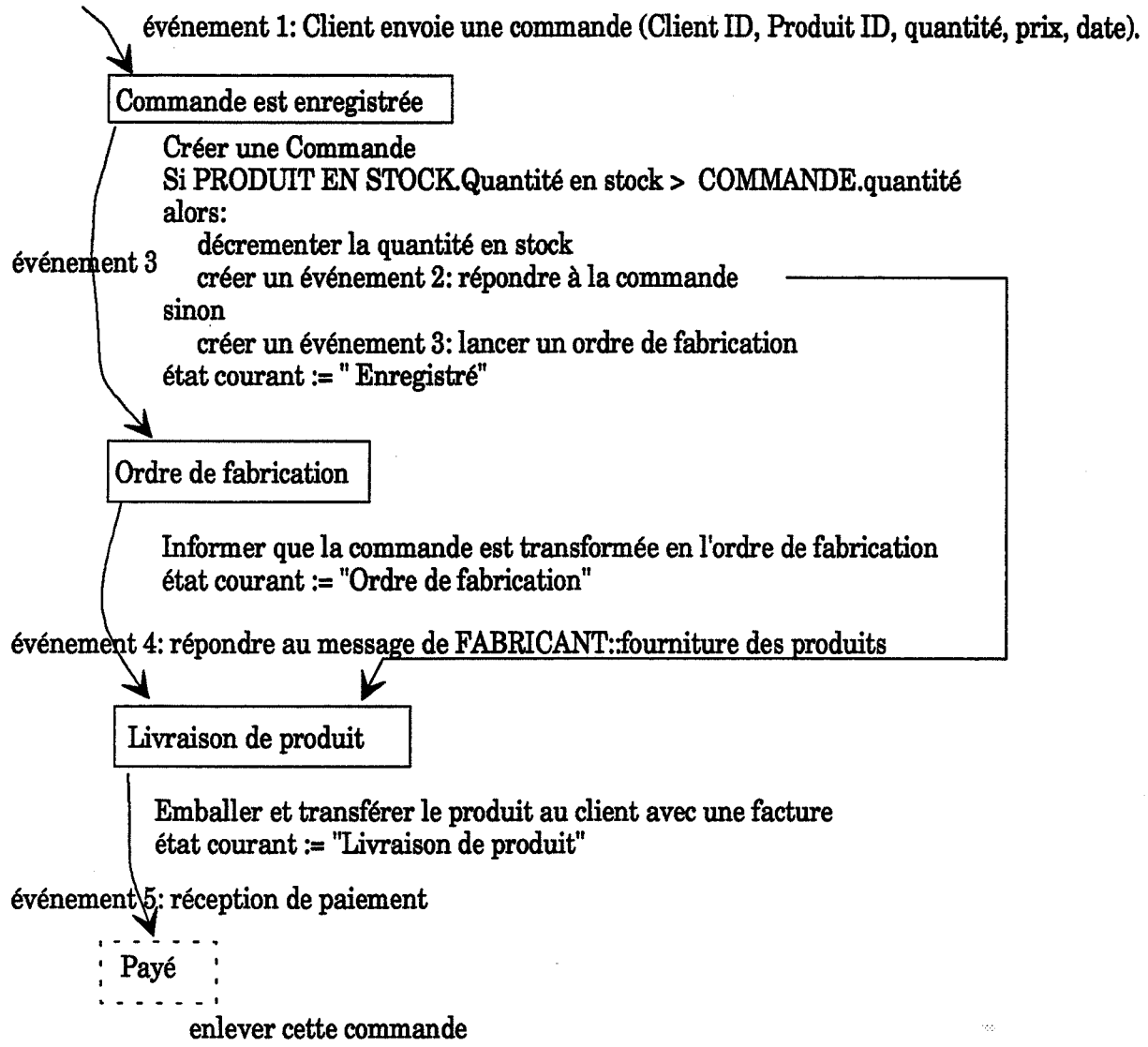


Figure 3-10 Comportement de l'Objet Interaction "Commande"

Quand ces objets sont identifiés, ils doivent être mis dans le cahier des charges à l'aide des quatre modèles suivants [SHLAER et al. 92]: des *modèles informationnels* qui décrivent les caractéristiques ou les attributs des objets et leurs relations qui persistent entre les objets, des *modèles d'états* qui décrivent la transition d'état ou le comportement temporel des objets, des *modèles de communication* qui décrivent la dynamique ou le contrôle global d'objets (les échanges d'informations internes et avec le monde extérieur) et des *modèles processus* qui décrivent différentes opérations d'une activité ou d'un événement. Nous discuterons les trois

premiers modèles dans la section suivante, et le quatrième sera présenté à la phase de conception des classes (ou plutôt de conception des méthodes concurrentes).

III Analyse de l'Application

L'analyse de l'application prend les modèles de l'espace de problème, créés pendant la phase d'analyse du domaine, comme des entrées et se concentre sur l'application courante à construire afin de transformer les modèles du domaine en modèles d'application. Les exigences des clients sont utilisées comme des contraintes qui rétrécissent la quantité d'informations du domaine. Les informations retenues pour une application particulière sont donc influencées par une grande vue de l'analyse du domaine. La structure d'application obtenue est plus générale et robuste que celle qui est développée directement à partir de cahier des charges d'une application particulière à construire.

Les modèles créés pendant l'analyse du domaine représentent des concepts importants du domaine sans tenir compte des cas particuliers d'application ni des représentations informatiques. L'analyse de l'application filtre les informations du domaine et impose des conditions qui accompagnent les contraintes informatiques (logicielles ou matérielles). Deux extensions doivent être considérées: l'enrichissement des classes d'objets vers les applications et la concrétisation de différents modèles d'application (figure 3-11).

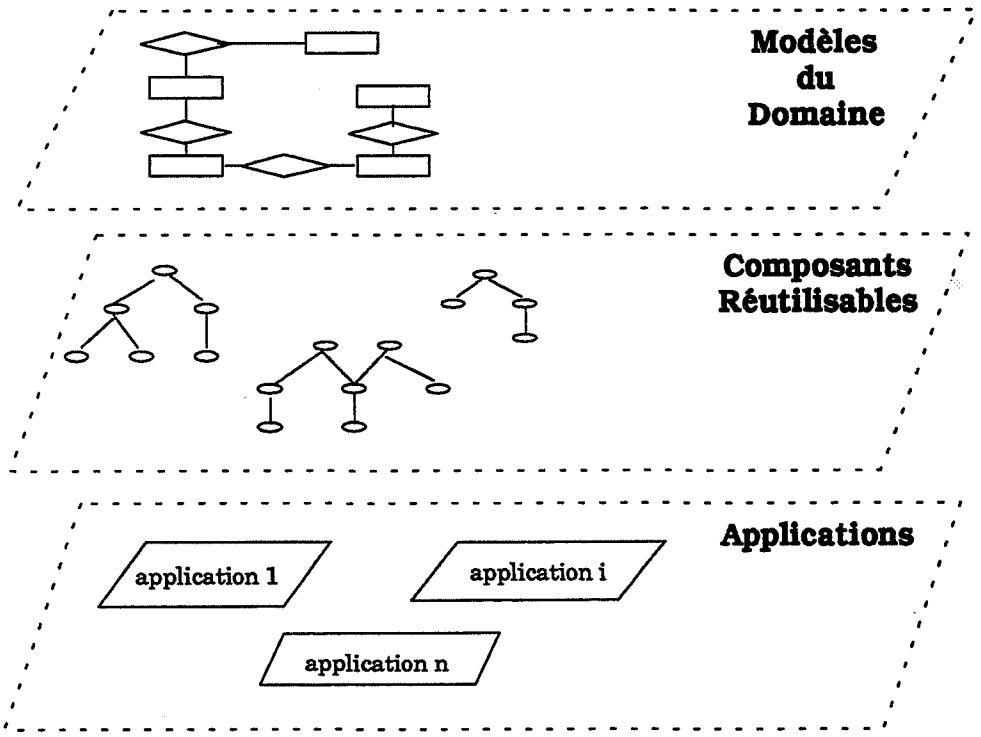


Figure 3-11 Etapes d'Analyse Orientée-Objets

III.1 Spécification des Classes d'Objets pour l'Application

Une application est un cas particulier du domaine. Elle est en général un problème complexe. La complexité tient à la nature et au nombre d'éléments à modéliser, à la diversité et au nombre d'attributs à créer, à la sophistication et à la quantité des règles de gestion. Il est impossible de construire un modèle général et rigoureux simultanément. Nous essayons ici d'établir un modèle des systèmes de production qui est assez général et, qui s'adapte à la simulation en utilisant l'approche par objet.

Pour construire une telle application, il ne suffit plus de faire une analyse du fonctionnement de chacun de ses éléments avant de les assembler. Il faut spécifier les comportements de chacun élément (complication de l'élément) d'une application avant d'affecter un rôle à chacun de ses éléments dans la phase de conception d'une application. Nous avons identifié cinq catégories d'objets dans la phase du domaine. Ces objets doivent d'être concrétisé et spécifiés par différents modèles graphiques et textuels dans un cahier des charges: modèle de communication, modèle des transitions d'état, modèle informationnel, modèle de processus, pour qu'un utilisateur puisse les instancie et concrétise dans son modèle de l'application. Nous les montrons les spécifications de ces modèles à travers quelques exemples d'objets physiques des systèmes de production.

III.1.1 Modèles de Communication de Classes d'Objets

Un système de production saisit les ordres de fabrication, transmet ses ordres sur diverses machines qui effectuent les opérations associées et délivre les produits finis aux clients. La tâche d'une machine, par exemple, est représentée par le processus suivant:

```
Processus Machine::fonctionner
Répéter
    Recevoir une commande (précisement un article)
    Transformer l'article
    Expédier l'article
A l'infini
```

Pour un atelier constitué de n machines et donc modélisé par n processus concurrents, il n'est pas évident de synchroniser ces n processus (avec un ou plusieurs processeurs) par les méthodes procédurales. Par contre, la notion de processus permet de maîtriser la complexité d'un système concurrent. Chaque processus peut être comparé à un programme séquentiel: il exécute des instructions, l'une après l'autre. La juxtaposition de plusieurs processus va

toutefois permettre d'exprimer des activités qui ne sont pas séquentielles. Les interactions des processus entre les objets peuvent être spécifiées à travers leurs modèles de communication.

Le modèle de communication d'une machine avec le stock amont à capacité infinie, les ressources qu'elle utilise, le stock aval à capacité infinie, par exemple, peut être schématisé comme dans la figure 3-12. La machine prend un article dans le stock amont. Si le stock est vide, la machine est bloquée; sinon, elle acquiert les ressources associées et transforme cet article. Dans le cas de stock vide ou de ressources associées non disponibles, cette machine est bloquée, et va être réactivée par deux messages extérieurs: un article vient d'être alimenté dans le stock amont vide ou des ressources sont redevenues disponibles. Après la transformation de l'article, la machine expédie cet article dans le stock aval et libère les ressources associées. Si le stock aval est vide ou les ressources associées réclamées antérieurement non disponibles, elle va envoyer un message à la machine aval qui consomme dans ce stock ou à la machine bloquée par les ressources pour la débloquer. Ce diagramme fournit une récapitulation graphique du comportement temporel de la machine et les interactions avec les entités extérieures. Ce modèle graphique a plusieurs objectifs:

- décrire le contexte (environnement) dans lequel l'objet fonctionne;
- fournir un support pour identifier les états élémentaires de l'objet;
- fournir un support pour définir le cycle de vie de l'objet;
- identifier les méthodes membres de l'objet;
- identifier les messages potentiels envoyés ou reçus par cet objet;
- servir comme moyen de communication entre les différents intervenants: chefs du projet, analystes, concepteurs, implémenteurs, etc.

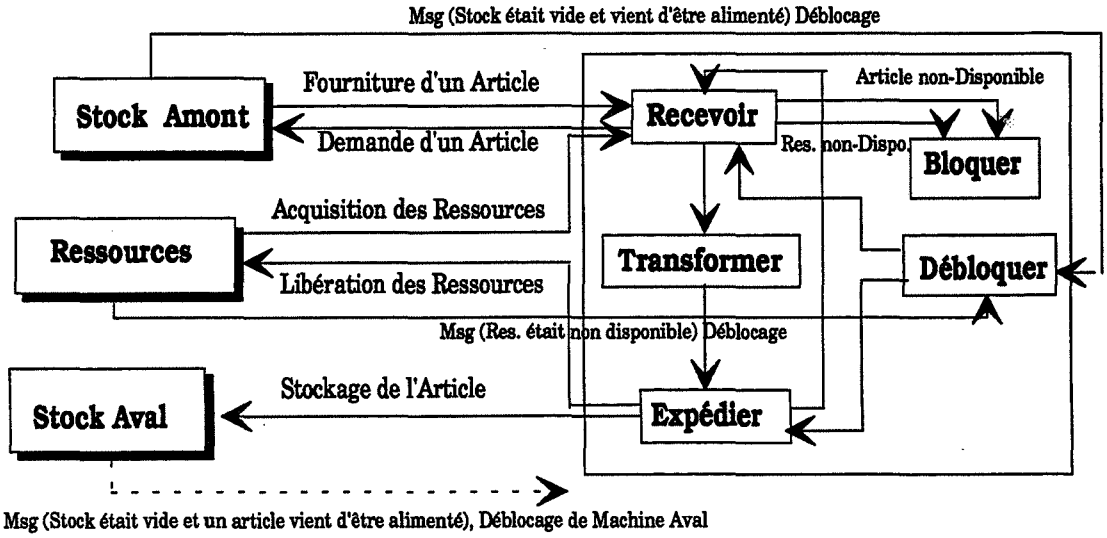


Figure 3-12 Modèle de Communication d'une Machine avec Deux Stocks à Capacité Infinie

Si plusieurs machines sont identiques ou si elles ont le même comportement au sens technologique, on peut les regrouper en une station pour l'optimisation de l'utilisation de la ressource processeur et l'utilisation des concepts objets tels que l'abstraction et l'encapsulation. Une station est un objet processus constitué de n machines, appelées serveurs, qui ont le même comportement (méthode *fonctionner*). Chaque station a une file d'attente amont et une file d'attente aval communes aux n serveurs (figure 3-13).

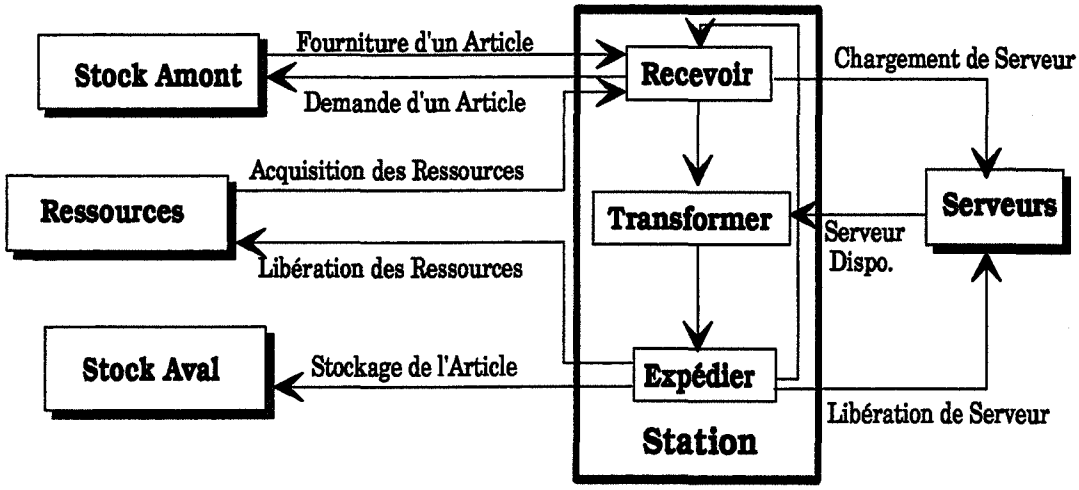


Figure 3-13 Modèle de Communication d'une Station (Niveau Fonctionnel)

Les processus de la station peuvent être modélisés de deux façons:

- **un processus par serveur**: la station est définie comme une interface qui gère les flux des articles entre les serveurs. Elle n'intervient pas directement sur l'horloge du système. On dit souvent que cet objet possède n processus en parallèle [ROSE 92]. La notion de processus est utilisée pour définir des méthodes concurrentes de l'objet station.

- **un processus pour tous les serveurs**: l'objet station prend la responsabilité de charger et décharger les serveurs qui lui sont associés. Le serveur n'est plus un processus; il devient un objet passif qui simule des opérations d'une machine. Il peut être en état libre, en état arrêt structurel, en état productif, etc. Le comportement de la station est conditionné par les états de ses serveurs, des stocks amont et aval, et la disponibilité de ressources associées aux serveurs. Le comportement de la station est très similaire à celui d'une machine et ne s'en distingue que par son interaction avec le noyau de synchronisation du système:

Processus Station::fonctionner
Répéter
  Charger les serveurs en état libre
  Transformer les articles chargés
  Expédier les articles transformés
A l'infini

Les serveurs travaillent en parallèle, éventuellement à différents rythmes. Pour simuler ces n processus en parallèles, l'avancement de l'horloge du système par le processus de station est différent de celui d'une machine (NB: dans le cas d'une simulation à événements discrets); la station cherche la fin d'activité du serveur le plus proche, calcule cette durée de transformation et avance le temps d'horloge du système de cette durée. Après la transformation, la station décharge les articles transformés et charge les serveurs qui viennent d'être libérés. Ce processus ne se bloque que s'il n'y a plus d'articles dans le stock amont et que tous les serveurs sont en état *libres* (figure 3-14) ou si le stock aval est plein et que tous les serveurs sont en état *déchargement*. La station peut être arrêtée pour plusieurs raisons; et elle doit être redémarrée quand les conditions de fonctionnement sont remplies. Notons que dans la figure 3-14 des processus d'une station, pour simplifier la représentation, nous n'avons pas intégré les processus *bloquer* et *débloquer* de la station.

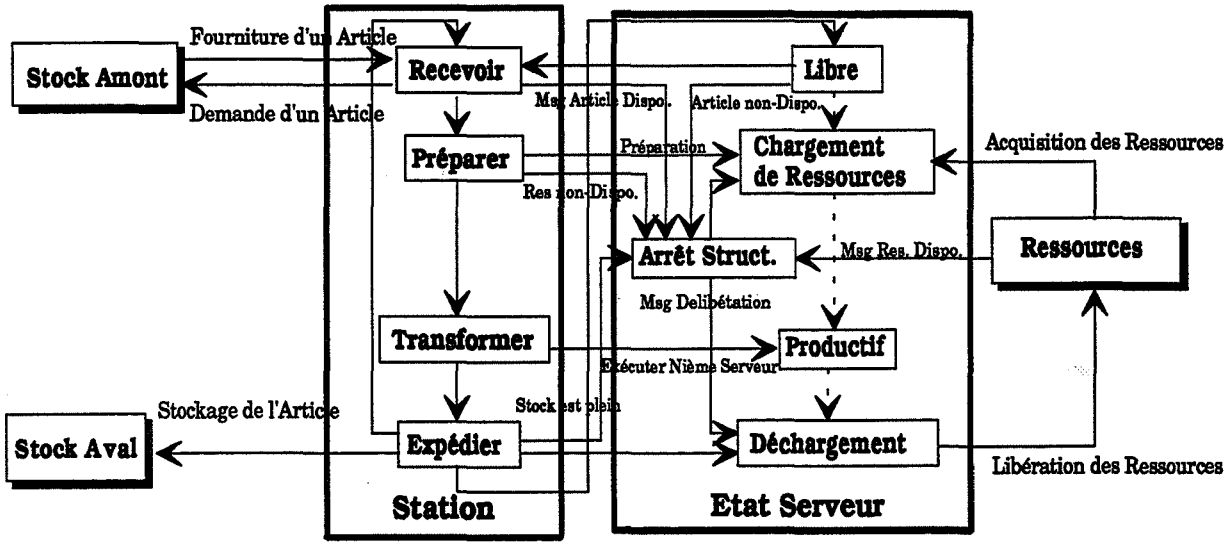


Figure 3-14 Modèle de Communication d'une Station

Les moyens de transport comprennent plusieurs catégories: les convoyeurs, les transporteurs guidés, les transporteurs libres, etc. Un transporteur, en incluant les liens avec le stock amont qui prend des pièces, le stock aval qui dépose des pièces et les ressources qu'il utilise, doit communiquer avec une autre entité réseau pour choisir le trajet de transport. Les processus (ou

activités) principaux d'un transporteur sont: charger des pièces, transporter des pièces, décharger des pièces, déplacer à vide, bloquer ou débloquer une activité, etc. Le modèle de communication est illustré dans la figure 3-15. La gestion de message est plus compliquée que celle de la machine et de la station par le fait que l'attribut taille de lot de transport joue un rôle décisif et il peut être variable selon le type de pièces à transporter. Nous ne détaillons pas ici la circulation des messages.

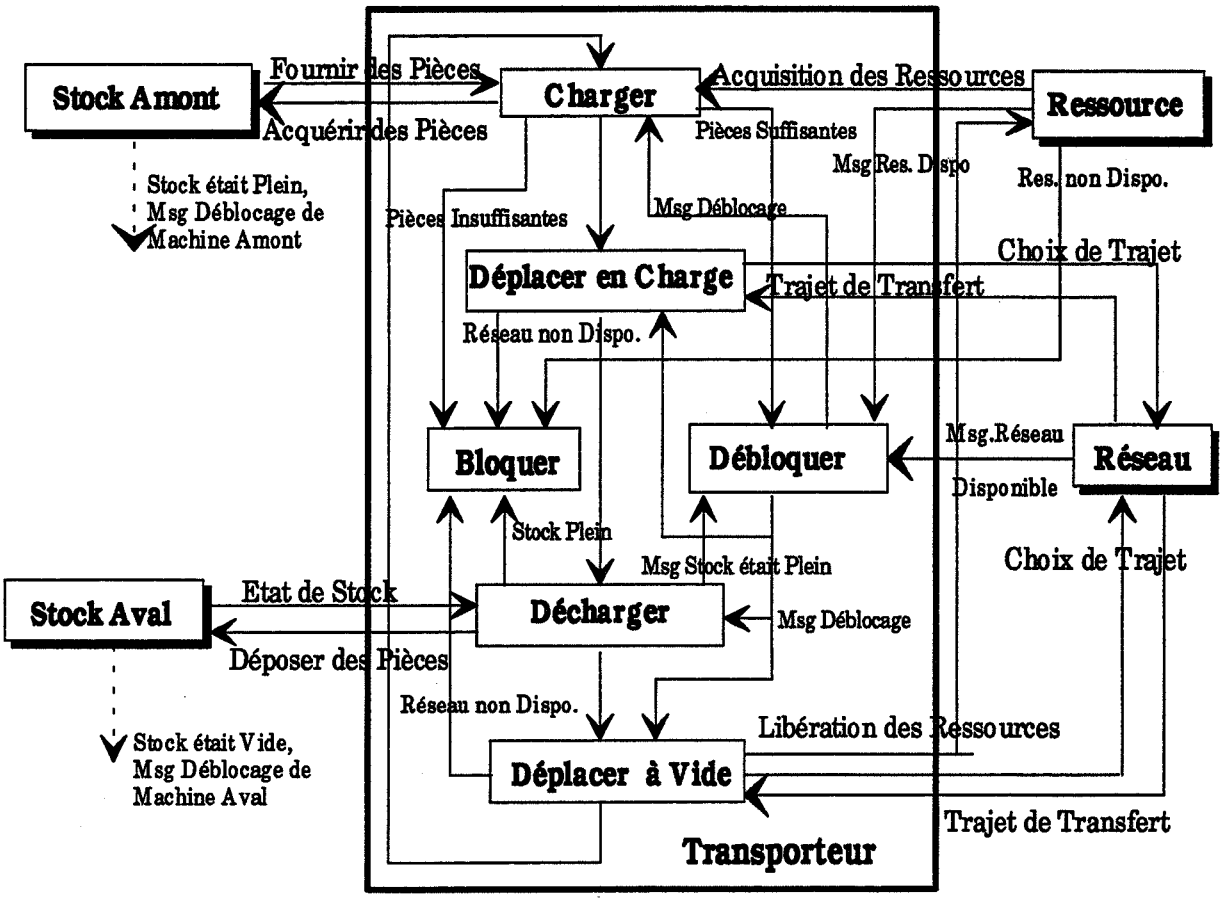


Figure 3-15 Modèle de Communication d'un Transporteur

Nous pouvons aussi compléter les modèles de communication pour toutes les entités identifiées dans la phase d'analyse du domaine. Par exemple, un autre diagramme de communication de machine avec des stocks d'entrée/sortie à capacité limitée (figure 3-16) est différent de celui d'une machine avec des stocks à capacité infinie. La capacité des stocks amont et aval de la machine (des objets extérieurs) modifie, comme on peut le constater dans la figure, le comportement (méthode *fonctionner*) de la machine.

Peu à peu, les classes d'objets du domaine et leurs comportements sont aussi définies. Remarquons que les modèles de communication sont construits au niveau du domaine, ils

peuvent être simplifiés ou enrichis dans une application particulière suivant la complexité de l'application, les objectifs de simulation et les rôles que les objets jouent dans l'application.

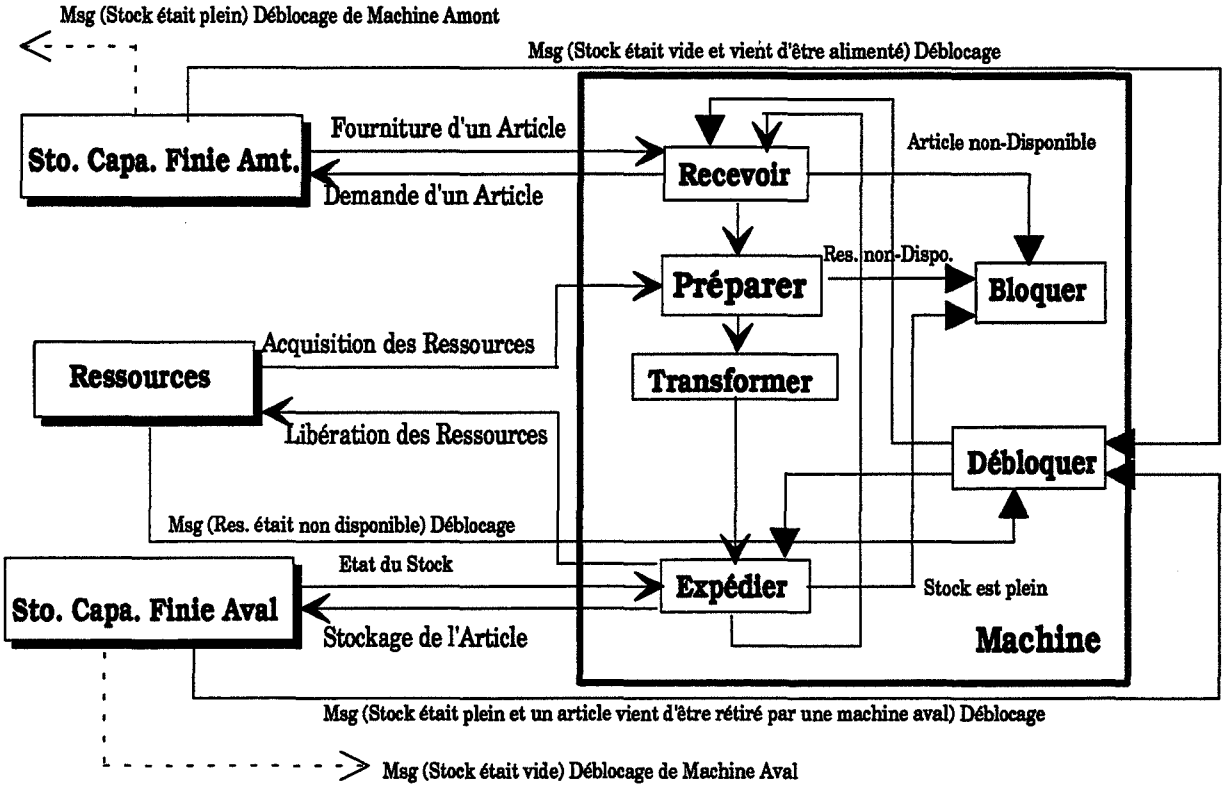


Figure 3-16 Modèle de Communication d'une Machine avec Deux Stocks à Capacité Finie

III.1.2 Modèles des Transitions d'Etat des Classes d'Objets

Lorsqu'on observe des objets dans un système de production, on constate qu'au cours du temps, ils passent successivement, de façon prévisible ou aléatoire, par un certain nombre de phases différentes, ou états élémentaires. Le modèle d'état sert à représenter le cycle de vie d'un objet ou le diagramme des transitions d'état d'un objet. On présent ici cet aspect à travers un exemple des transitions d'état d'une machine.

Lorsqu'une machine n'est pas mise en service, ou lorsqu'elle est décrétée indisponible pour des raisons de maintenance ou de modification, elle est à l'état "*non engagé*". Sinon, elle est à l'état "*engagé*", c'est-à-dire qu'elle est impliquée dans le processus de production. Elle contribue dès lors, qu'on le veille ou non, aux performances du système de production.

Une machine est susceptible, dès qu'elle se trouve libre, de recevoir un article à transformer. L'état "engagé" n'est donc pas toujours synonyme de "productif": il comporte en réalité plusieurs sous-états, dont un dit "productif" et plusieurs états "arrêts". Une machine est en état

productif lorsqu'elle est en opération normale sur le produit. Cette transformation peut être physique (c'est le cas général) ou logique, comme dans le cas d'une opération de test qui transforme un produit "à contrôler" en un produit "contrôlé". L'essentiel est que ce moyen soit effectivement en train de participer à l'évolution du produit dans le système de production, conformément à la mission pour laquelle elle est conçue.

Une machine peut être arrêtée pour plusieurs raisons. Elle peut être en état d'arrêt structurel et en état d'arrêt conjoncturel dans les quatre cas suivants.

1) *Blocage amont*: attente d'approvisionnement en composants. La machine se trouve en état de fonctionnement mais, pour des raisons qui lui sont extérieures, elle ne peut participer à la production. Elle se trouve en état *prêt-à-charger* (ou en état *disponible*) si on utilise les termes de simulation.

2) *Saturation en sortie*: attente d'évacuation des composants transformés. C'est le cas inverse du précédent, qui provoque le même effet de blocage de l'activité de la machine. Elle est en état *prêt-à-décharger*.

3) *Réglage pour le changement d'outillage ou de composant*. Ce cas d'arrêt structurel est dans une certaine mesure dû à la nature et à la conception de la machine mise en oeuvre pour une opération donnée. Une machine peut donc être en état préparation ou en état prêt à charger si la préparation est finie mais qu'il manque des composants en entrée. Il est fugitif et artificiel. En fin de préparation, la machine passe de l'état *préparation* à l'état *prêt-à-charger*.

4) *Panne ou maintenance*. Une machine peut subir des pannes de toute nature: mécanique, électrique, hydraulique, informatique, etc. Ces arrêts propres d'une machine induisent des arrêts subis par les produits en cours de fabrication et par les opérateurs en poste sur cette machine. On dit que la machine est en état de *panne* (remarquons que dans la figure 3-17 on ne représente que la panne non pré-emptible.). Elle peut aussi être arrêtée pour maintenance ou visite technique. Elle est alors en état *maintenance*.

Ces arrêts ou pertes de temps seront de nature et d'importance différentes selon les processus de production auxquels ils appartiennent. Ils sont liés à la nature de ces processus et aux individus (objets) impliqués, ou plus généralement à l'organisation de l'ensemble des moyens de production. C'est pourquoi il est intéressant de mettre ces arrêts en évidence, dès lors qu'ils consomment du temps et de l'argent sans faire progresser les articles (produits) dans les systèmes de production.

A partir des états ainsi énumérés, nous sommes en mesure de construire un diagramme de phase qui constitue un modèle des transitions d'état d'une machine (figure 3-17).

Si le but essentiel de l'activité de production est de faire évoluer les produits vers un état d'achèvement précis, cette évolution, au niveau élémentaire, nécessite la rencontre pendant un certain laps de temps entre un produit et une machine, un opérateur ou d'autres moyens de production. Cette rencontre, et ceci sera explicité dans la partie simulation par les différents états d'objets, s'opère au prix d'un certain coût opérationnel. C'est la prolifération plus ou moins contrôlée de ces rencontres (états) élémentaires qui constitue l'activité du système de production et c'est l'accumulation des coûts opérationnels élémentaires qui engendre le coût de production du produit (et non le prix de revient, qui intègre bien d'autres coûts).

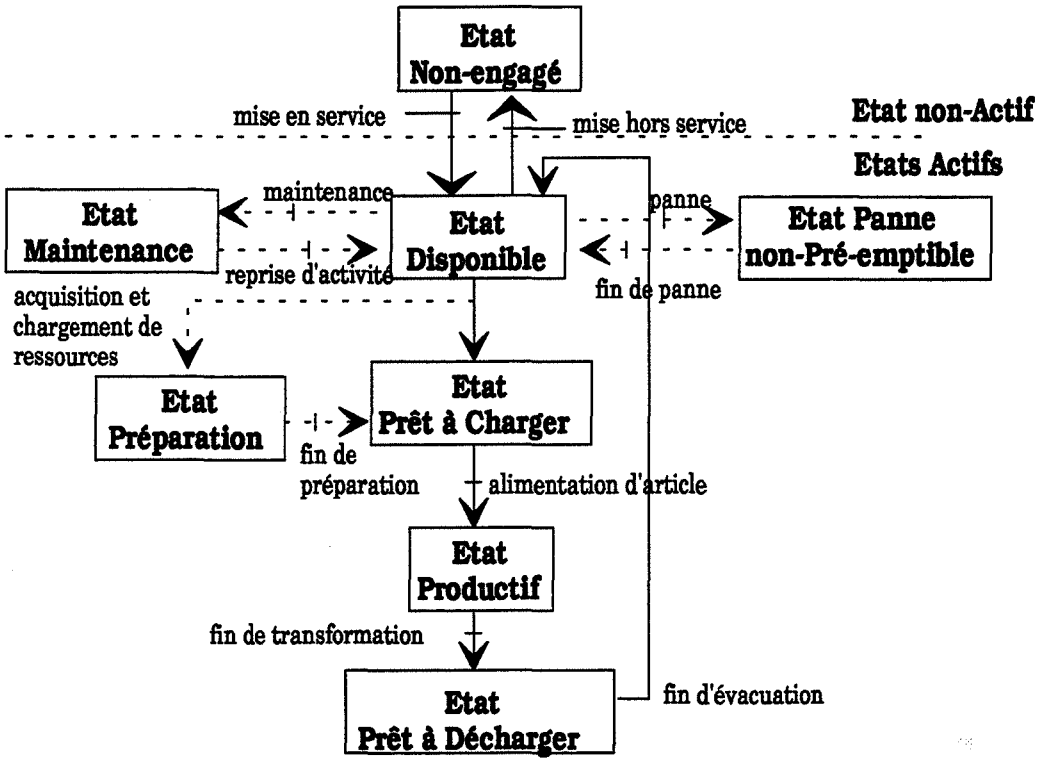


Figure 3-17 Modèle des Transitions d'Etat d'une Machine (Niveau du Domaine)

De la même manière, nous modélisons les modèles d'état des autres entités réelles. Une ressource utilisée par une machine, par exemple, peut être dans différents états (figure 3-18):

- Etat *non-engagé*: la ressource est hors de service;
- Etat *libre*: la ressource est mise en service, mais elle n'est pas encore prise par un objet processus (une machine, une station ou un transporteur);
- Etat *préparation*: la ressource est acquise par un processus qui est dans l'état préparation;
- Etat *productif*: la ressource est utilisée par un processus pour une opération sur une pièce;

- Etat *prêt-à-décharger*: la ressource est prête à être déchargée d'un processus;
- Etat *panne*: la ressource ne peut pas être utilisée par un processus

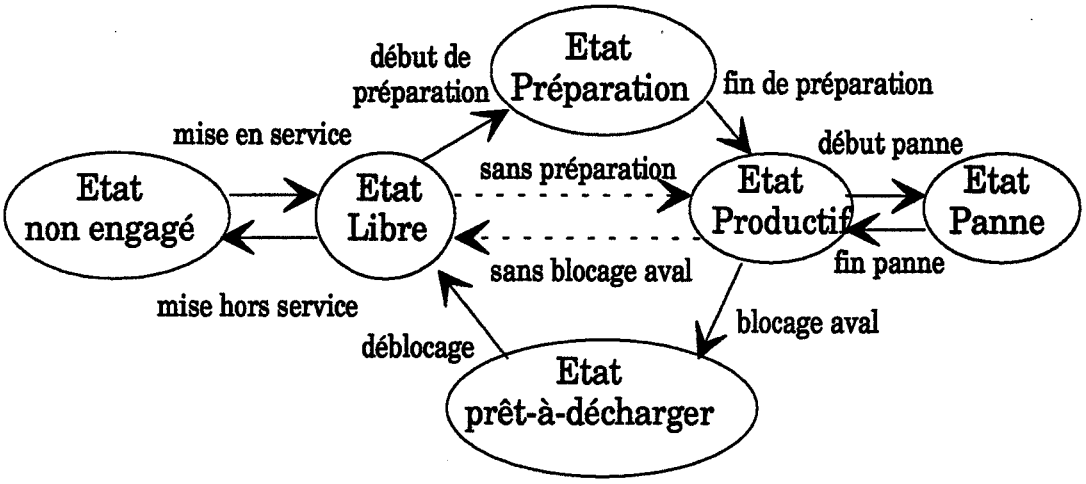


Figure 3-18 Diagramme des Transitions d'Etat d'une Ressource Partagée

Evidemment, une machine peut être spécialisée en des machines particulières comme un robot, une machine à commande numérique, etc. On peut construire, de la même manière le modèle d'état correspondant de ces machines [RODDE 90]. La figure 3-19 illustre un diagramme des transitions d'état d'une machine à commande numérique. Ces états d'objets conditionnent les comportements d'objets (l'enchaînement de leurs méthodes) et illustrent les modifications des attributs des objets concernés. Les performances du système de production sont représentées, dans l'approche orientée-objets, d'une part par les attributs des objets, d'autre part par les relations entre les objets. C'est le moment que nous aborderons les modèles informationnels des classes d'objets qui définissent les caractéristiques d'objets.

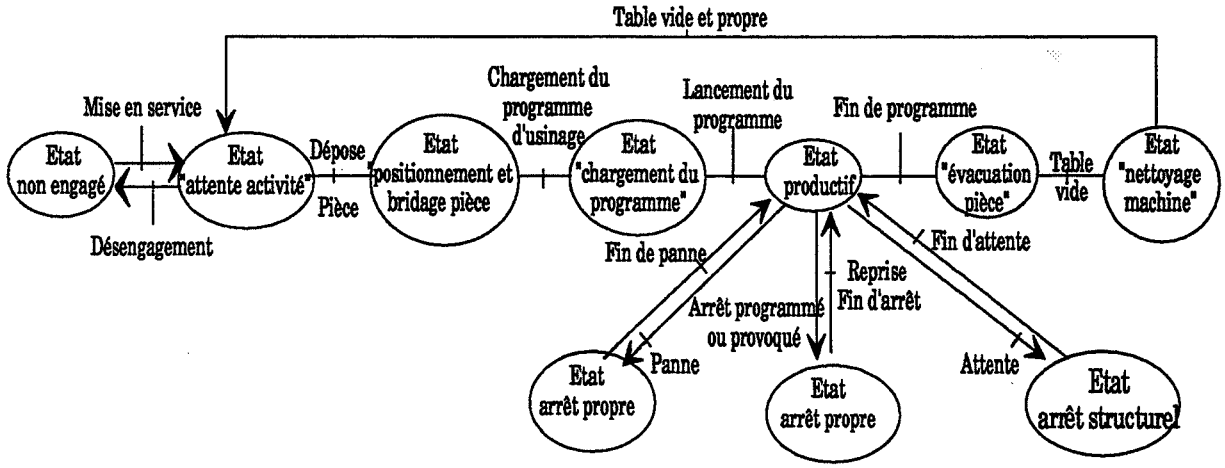


Figure 3-19 Diagramme des Transitions d'Etat d'une Machine à Commande Numérique

III.1.3 Modèles Informationnels des Classes d'Objets

Un modèle informationnel d'un objet décrit les caractéristiques ou les données essentielles de l'objet et les relations qui persistent entre cet objet et le monde extérieur. Une machine, par exemple, est caractérisée par ces données. En terme d'analyse orientée-objets, ces données sont appelées les attributs d'objets et elles sont représentées dans le modèle d'information d'objets. Ces attributs peuvent être différenciés en différentes catégories; nous prenons l'exemple de l'objet machine pour les illustrer.

a) Les *attributs structurels* caractérisent les aspects fondamentaux d'une machine. Certains d'entre eux décrivent les liens logiques entre les différents objets, et leur spécification fine dépend essentiellement de la nature des applications et des conditions d'exploitation. En ce qui concerne la simulation des flux de production, les principaux attributs structurels d'une machine comprennent: son identifiant, son coût unitaire d'utilisation, son opérateur associé, les références de composants à fabriquer, les files d'attente d'entrée/sortie et leurs caractéristiques (modes de gestion), son implantation géographique, etc. Ils expriment l'aspect intrinsèque ou organisationnel de la machine.

b) Les *attributs conjoncturels* expriment la disponibilité effective des objets ou le contexte dans lequel doit se dérouler la production. Ils sont, dans une large mesure, non modifiables dans le cadre du système de production lui-même et représentent plutôt les contraintes extérieures comme la prévision de non-engagement, le plan de charge.

c) Les *attributs instantanés* expriment l'état courant des objets. Ils rendent compte en temps réel des conditions d'utilisation de la machine. En ce qui concerne les attributs relationnels permettant de la situer dans le système de production, ils comprennent: référence du produit ou lot de produit en cours, compteur de pièce pour le lot en cours, etc. On n'aborde pas ici les attributs d'état intrinsèques, telles que les valeurs courantes des paramètres du fonctionnement (températures, pression, etc.), sauf la valeur courante de la machine pour générer des indicateurs de sortie dans le but de calcul économique.

d) Les *attributs événementiels* sont ceux qui provoquent l'évolution des attributs instantanés. Ce sont l'ensemble des phénomènes dont dépendent le modèle des transitions d'état de machine à un instant donné. Ces attributs principaux comprennent le temps opératoire, le temps de panne, le temps de retouche, le temps de réglage, le temps d'attente, etc.

e) Les *attributs historiques* sont ceux qui permettent de rendre compte de façon plus ou moins détaillée du passé récent de la machine comme le temps productif, le temps d'arrêts

propres (le cumul des temps de panne), le temps d'arrêts induits (le cumul des temps d'attente), le temps en état réel productif, le temps total observé, etc.

f) Les *attributs statistiques* sont le résultat de calculs permettant de compacter les attributs historiques sur de longues périodes, de manière à dégager les lois de fonctionnement du système de fabrication concerné. Ces calculs fournissent une image très représentative des capacités réelles d'une machine comme la loi sur la durée de panne, le taux de rebuts, le taux d'occupation, le nombre total d'ordres lancés et terminés sur une période donnée, etc.

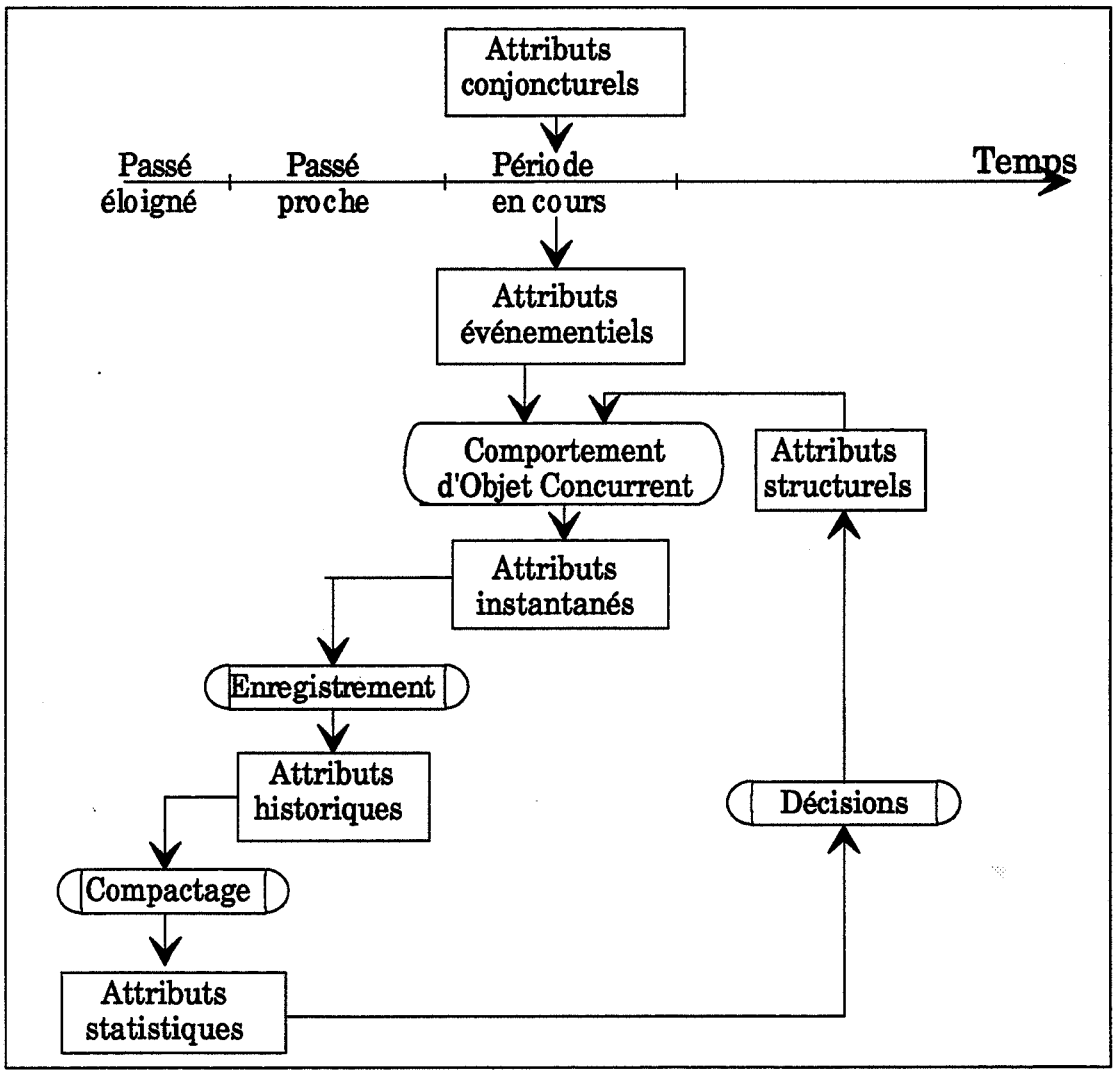


Figure 3-20 Relations entre les Attributs d'un Objet Concurrent

Bien entendu, aucune de ces catégories d'attributs n'est *a priori* figée. Les attributs structurels eux-mêmes évoluent en fonction des décisions prises au vu des résultats et performances du système de production; autrement dit, suivant les attributs statistiques. Certains attributs sont définis aussi comme des méthodes d'après le détail de la modélisation et de la simulation. Tous

les attributs évoluent au cours de la production sous l'effet du comportement des moyens de production (figure 3-20) sauf les attributs conjoncturels qui ont une influence sur l'atelier sans recevoir de réactions en retour (informations entrées dans les constructeurs d'objets).

Une machine peut être regardée sous différents *points de vue*: organisationnels, processus (transition d'état), flux de production, agrégation/désagrégation au sens de la modélisation, niveaux de décision, etc. Au point de vue processus, une machine possède un cycle infini qui acquiert des articles dans le stock amont, les transforme et les expédie vers le stock aval (voir le modèle de communication). La méthode *fonctionner* de la machine définit ses différentes transitions d'état possibles, les processus impliqués dans ces transitions et les coordinations temporelles et spatiales avec d'autres objets. Si nous encapsulons ses activités, une machine peut être vue comme une boite noire qui transforme des flux d'entrée (FAE) en flux de sortie (FAS). Sous un point de vue organisationnel, une machine a une file d'attente en amont et une file d'attente en aval (figure 3-21). Elle peut posséder un pointeur vers l'objet "Panne" (un autre processus) pour son traitement de pannes, un pointeur vers l'objet "Palette" pour modéliser son temps de préparation, un pointeur vers l'objet "Inspecteur" pour représenter le contrôle de la qualité de transformation sur des articles. Elle peut interagir avec le contexte extérieur par les messages "*Ordres*". L'ensemble de ces objets (agrégation) interagissent et l'enchaînement des activités de ces objets sont définis par le comportement (la méthode *fonctionner*) de la machine.

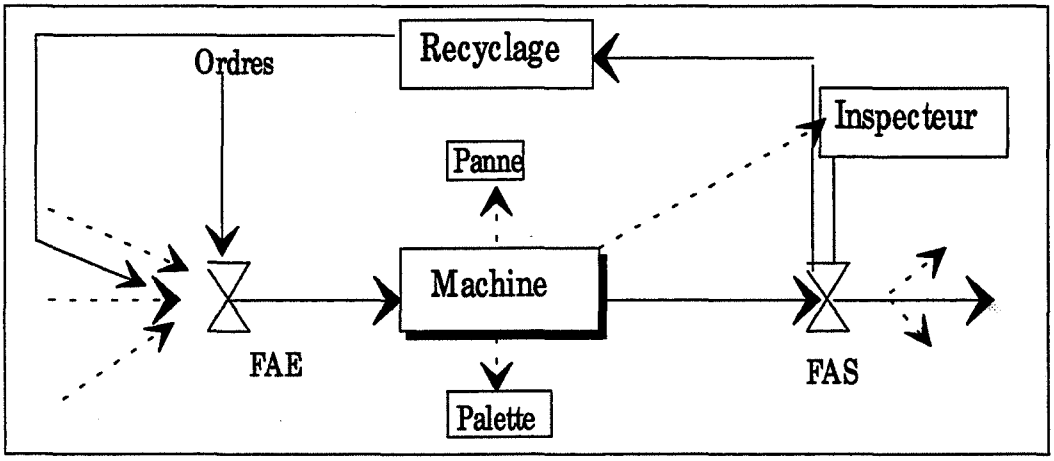


Figure 3-21 Structure Générale d'une Machine

L'intérêt de considérer un objet machine sous ces différents points de vue est d'identifier sa structure (les attributs) et les opérations associées afin de mieux définir les attributs et les services (fonctionnalités) de la machine en terme objets (figure 3-22). Nous concrétisons des services identifiés à la phase d'analyse en des méthodes d'objets à la phase de conception par objet dans les deux chapitres suivants.

Identifiant	état
capacité	référence de pièce ou lot de pièce
coût unitaire d'utilisation	compteur de pièce pour le lot en cours;
implantation géographique	
opérateur associé	valeur
type de pièces compatible avec l'équipement	temps de panne
files d'attente d'entrée (FAE)	temps de retouche
files d'attente de sortie (FAS)	temps de réglage
	temps d'attente
caratéristiques des FAE	temps d'usinage
caractéristiques des FAS	
prévision de non engagement de machine	taux de rebuts
plan de charge	taux d'occupation
	loi sur la durée de panne (MBTF)
temps productif	nombre d'ordres (lots) lancés
temps d'arrêts propre(cumul de temps de panne)	nombre d'ordres (lots) terminés
temps d'arrêts induits(cumul de temps d'attente)	
temps observé	
	ordonnancer cette machine
mise en service	bloquer la machine
définr une règle de priorité	débloquer la machine
sélectionner une opération	traiter la panne
saisir une pièce	visite technique
acquérir des ressources	mise à jour des attributs
transformer la pièce	mémoriser les valeurs d'attributs
expédier la pièce	calculer le taux d'utilisation
libérer les ressouces	méthodes statistiques
stocker la pièce	etc.

Figure 3-22 Attributs et Fonctionnalités d'une Machine

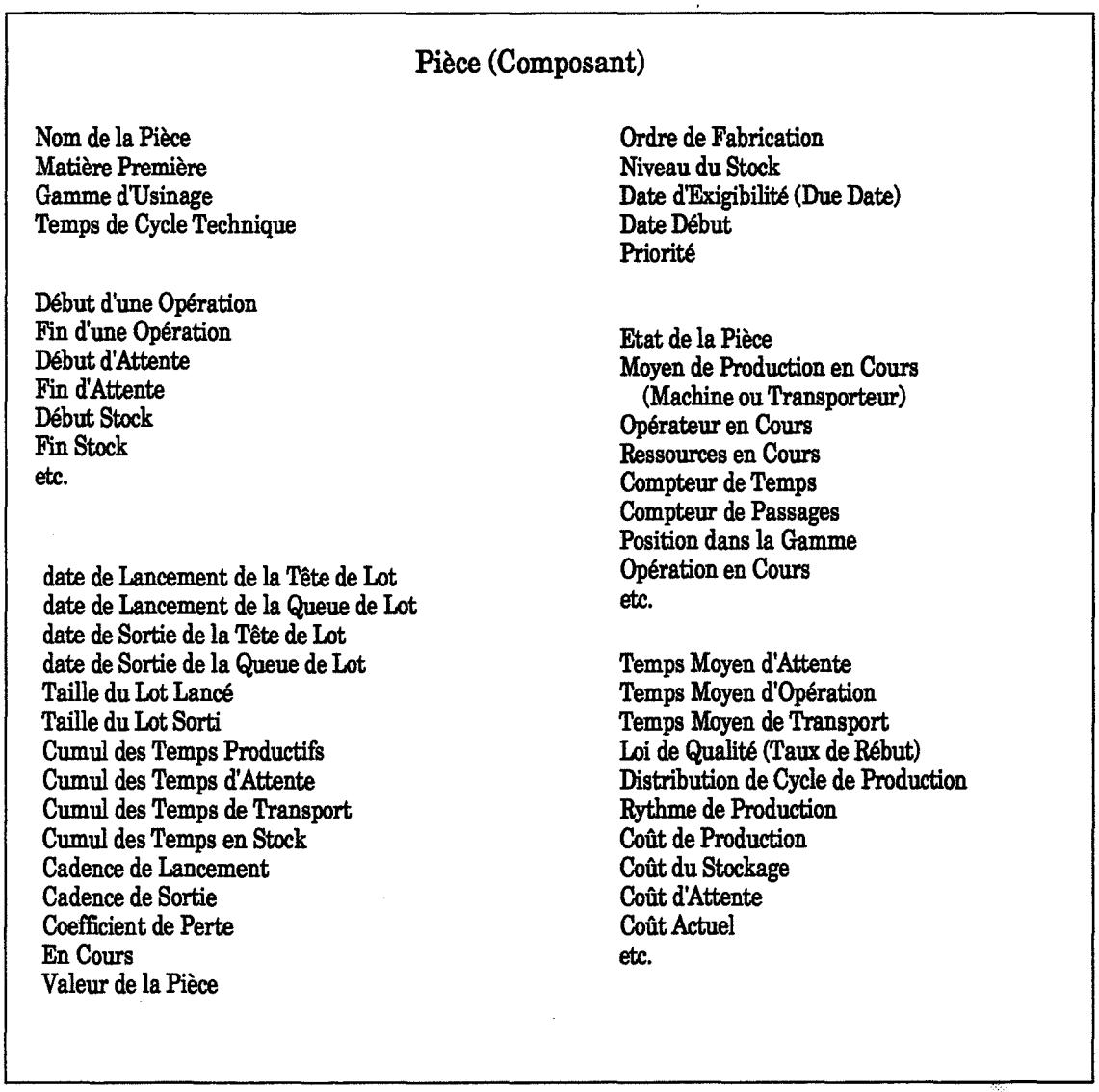
De la même manière, on peut identifier les attributs des autres entités dans les systèmes de production. Un article ou une pièce qui circule dans les systèmes de production, par exemple, possède aussi six sortes d'attributs comme ceux d'une machine (figure 3-23).

a) Les attributs *structurels* caractérisent les aspects technologies de la pièce. L'attribut le plus fondamental concernant la pièce est évidemment la gamme de fabrication. En ce qui concerne la simulation des flux de production, les principaux autres attributs structuraux comprennent: identificateur (nom), matière première, temps de cycle technique, etc.

b) Les attributs *conjoncturels* expriment les demandes extérieures et les ressources disponibles pour une unité de production tels que: ordre de fabrication, en-cours, priorité, date d'exigibilité, etc.

c) Les attributs *instantanés* expriment l'état d'avancement de la pièce par rapport à sa gamme. Le but recherché en général est que le temps de transit effectif de la pièce dans le

système de production soit le plus proche possible de son temps de cycle technique. Ils comprennent: opération et moyens de production en cours, position dans la gamme, état de pièce, etc.



Figue 3-23 Attributs d'une Pièce pour la Simulation des Flux de Production

d Les attributs *événementiels* expriment l'évolution des attributs instantanés. Ces valeurs vont permettre de déterminer avec précision comment la pièce a occupé le temps pendant lequel elle a fait partie du système de production: début et fin d'une opération, début et fin d'attente, début et fin de contrôle de qualité, etc.

e) Les attributs *historiques*: il est intéressant (et parfois obligatoire) de connaître dans quelles conditions une pièce a été élaborée dans le but de dégager les lois caractéristiques du

système de production. Ces attributs caractérisent plutôt la répartition des temps passés par la pièce dans le processus de production.

f) Les attributs *statistiques* rendent compte des lois représentatives du flux à travers le système de production. Ces données peuvent, d'une part servir de référence pour les résultats de simulation destinée à tester les chances de succès de telle modification, de tel ou tel scénario, ou les conséquences probables d'une décision ou d'une évolution conjoncturelle, d'autre part, devenir la matière première d'une comptabilité analytique fondée sur la connaissance objective de la réalité.

Les attributs des classes d'objets issues de la phase de l'analyse du domaine comme: station, transporteur, atelier, opérateur, produit, marché, client, fournisseur, sous-traitant, prévision, commande, achat, plan industriel, programme directeur de production, nomenclature, carnet de commande et de lancement, lot, gamme de fabrication, opération, etc., peuvent être ainsi identifiés et structurés. Ces trois modèles constituent un cahier des charges du domaine. Pour une application particulière, nous pouvons instancier ou étendre ces classes d'objets existantes, ou créer de nouvelles classes d'objets s'il n'existe pas de classes similaires. Notons que le cycle de vie de développement des classes est différent de celui du développent de l'application.

III.2 Construction des Modèles de l'Application

Une fois les entités du domaine et leurs fonctionnalités (les attributs et les services qu'elles doivent fournir) identifiées, nous devons les assembler et les mettre en ordre pour construire un modèle d'application. Les classes d'objets sont des moyens de morceler et de structurer une application pour la rendre modulaire et réutilisable, etc. Souvent, un ensemble de classes collaborent pour accomplir un but commun: ces classes constituent un *sous-système* ou une *sous-architecture* de l'application [MONARCHI et al. 92]. Un bon sous-système fournit un guide et une direction pour la conception et le développement d'une application. Ces sous-systèmes sont des entités conceptuelles (qui n'existent pas pendant l'exécution du programme). De la même manière, chaque sous-système est composé de sous-systèmes. C'est un modèle imbriqué. Chaque sous-système est caractérisé par une interface vers l'extérieur. Les sous-systèmes principaux sont pilotés par un module principal (dans le cas de simulation à objets, il s'agit d'un noyau de synchronisation qui pilote l'ensemble des objets actifs ou des processus parallèles) qui a une relation logique différente des relations d'héritage et des relations d'agrégation/désagrégation. Notons que la conception de classes d'objets a un impact majeur sur la facilité, la rapidité et la qualité du développement d'une application.

Une entreprise est un système complexe qui comprend plusieurs niveaux d'abstractions. Dans un système d'aide à la décision comprenant un module de simulation, plusieurs sous-systèmes existent: la production, l'environnement, la décision et l'évaluation, etc.

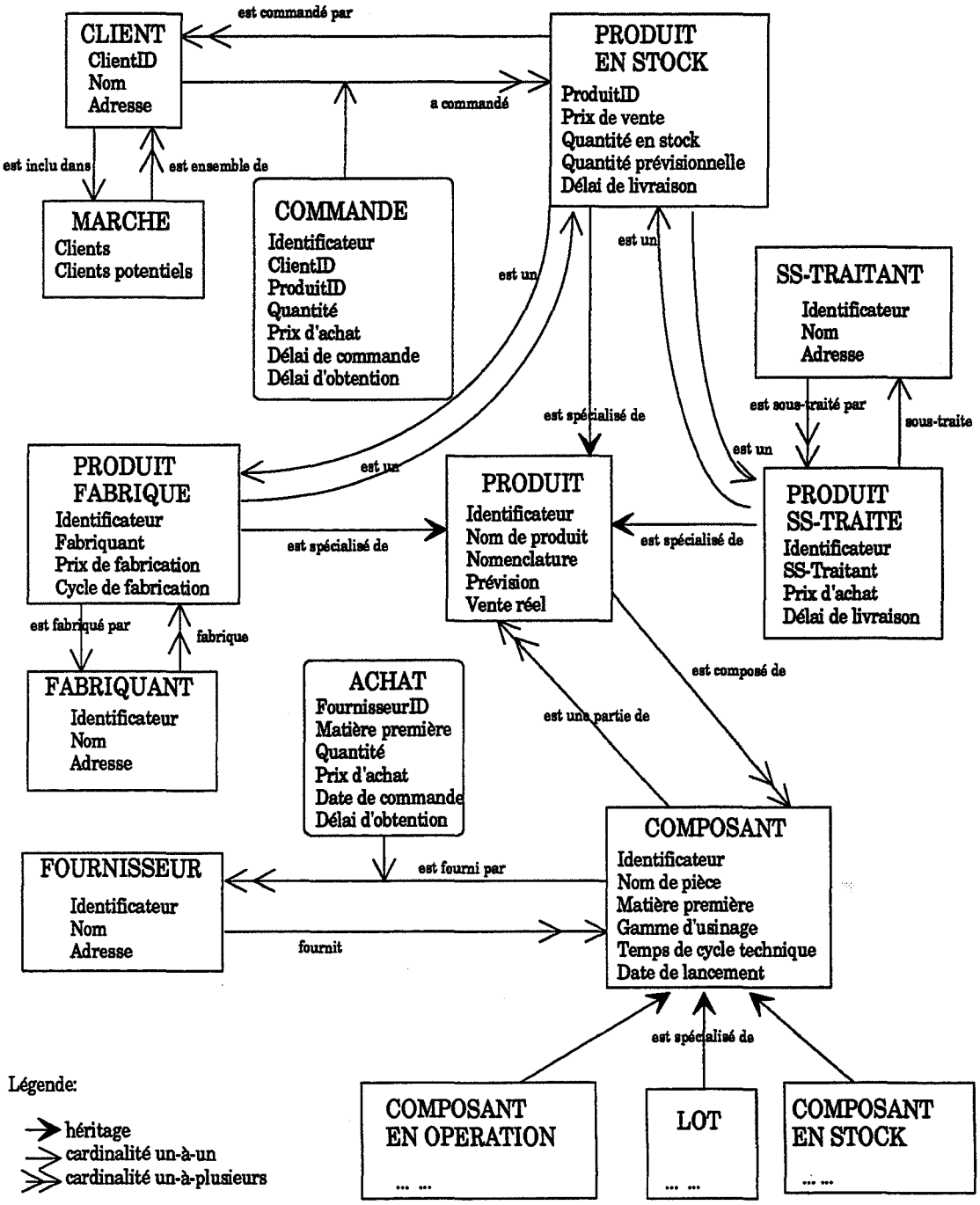


Figure 3-24 Contexte du Système de Production

Le *sous-système environnement* d'une entreprise comprend en général les fournisseurs, le marché, les clients, les sous-traitants et les produits commandés par les clients. Il définit le contexte dans lequel la production se déroule. Un objet marché est un ensemble de clients qui commandent ou peuvent commander des produits. Les classes d'objets décisionnels possèdent des méthodes pour ces décisions commerciales qui, selon l'état du système (capacité, suivi, finance, etc.), les commandes fermes et les fournisseurs, déterminent la quantité de produits à fabriquer. Un objet commande est donc une association qui a son propre comportement (figure 3-10). Les matières premières à acheter (approvisionnements) sont générées en suivant ces commandes, les états des produits et des composants en stock. Un objet achat est donc une autre association qui est caractérisée par les attributs: les composants ou les matières premières à acheter, les fournisseurs que l'on peut choisir, la quantité et le prix des matières à acheter, et la date d'obtention. L'architecture des ces classes d'objets est caractérisée par leurs relations logiques. C'est à la phase d'implémentation d'une application que l'on précise exactement les interactions. Les modèles conceptuels comme entité-association (figure 3-24) sont souvent utilisés pour représenter les résultats de l'analyse et de la conception d'application.

Le *sous-système décision* est le plus difficile à modéliser. Les méthodes traditionnelles sont orientées vers une décomposition fonctionnelle: diviser le problème en une séquence de tâches formant les blocs essentiels pour une application [GERSHWIN 89], [MCPHERSON et al. 86], [O'GRADY 86] (figure 3-25). Puisque ces entités de décision ou classes d'objets décisionnels (objets rôles ou spécifications) sont caractérisées par leurs services et non par leur représentation (d'ailleurs, la variabilité des méthodes de conception par objet et des langages à objets influe fortement sur la conception de ces classes), nous ne construisons que des classes d'objets de base la plus générale possible afin de construire des classes plus riches et plus sémantiques par les utilisateurs futurs pour ses applications particulières et donnons des guides essentiels pour la conception de ces classes décisionnels.

Deux sortes de décisions doivent être modélisées: des décisions *implicites* et des décisions *explicites*. A la fin d'une opération sur une machine par exemple, une décision implicite consiste à envoyer cet article traité vers une destination donnée et à acquérir un article suivant., dans ce cas là, la machine consulte l'état d'un objet article concerné par l'envoi d'un message à ce dernier (au point de vue de la relation agrégation) pour obtenir le moyen de l'opération suivante sur l'article et envoie cet article dans la file d'attente correspondante. Ces décisions sont réalisées par le mécanisme d'héritage et par des relations entre les composants internes (ou précisément les relations d'agrégation/désagrégation). Les concepts comme les classes abstraites, les fonctions virtuelles, ou les liaisons statiques et dynamiques facilitent, techniquement parlant, ces implémentations.

Les décisions explicites comme la décision d'achat, la décision de lancement, la décision d'approvisionnement, les maintenances, etc., sont très subjectives. Elles dépendent des outils et des langages de programmation utilisés. Nous les modélisons par deux approches. La première façon est d'instaurer une interface de dialogue avec l'utilisateur (*guide d'utilisateur* ou surveillance, chapitre IV) ([MIRY et al. 92], [MIRY 93], [VINCENT et al. 92]). La deuxième est d'envoyer un message proprement dit à un autre objet (*commande automatique*).

Normalement, cette méthode est virtuelle et abstraite, il faut redéfinir quand l'utilisateur instancie ces classes d'objets pour son application. Ces classes doivent comprendre: décision d'achat, décision commerciale, décision de prévision, décision d'investissement, décision de lancement, etc. Ces décisions sont soit noyées dans les méthodes (décisions statiques) des autres classes d'objets, soit des messages (décisions dynamiques) qui font des liens logiques entre des classes d'objets. Le développement de ces décisions explicites n'est pas un processus répétitif.

Le *sous-système évaluation* est un module d'analyse qui exploite des données (indicateurs) obtenues par la simulation pour évaluer les performances du système (MIRY et al 91], [MIRY et al. 93]). Plusieurs progiciels existant sur le marché (ou au laboratoire), permettent de construire des classes d'objets d'évaluation (histogramme, diagramme de Gant, objets d'affichage, etc.).

Le *sous-système production* consiste à construire un modèle de simulation des systèmes de production. Un système de production est un ensemble de moyens qui permet une transformation des matières premières en produits finis. Il comprend trois sortes d'entités à modéliser ([KELLER 92], [YE et al. 94b]).

1) Un sous-ensemble d'entités de transactions composé des matières premières, des produits semi-ouvrés, des composants à fabriquer, des nomenclatures, des gammes associées aux composants ou aux produits, des opérations, ainsi que des ordres de production ou de fabrication que le système de production peuvent réaliser. Ces entités de transactions peuvent être des objets physiques ou logiques (objets rôles ou spécifications).

2) Un sous-ensemble d'entités physiques défini l'ensemble des moyens de production et de manutention, leur répartition géographique et leurs interconnexions logiques et physiques (objets physiques ou spécifications).

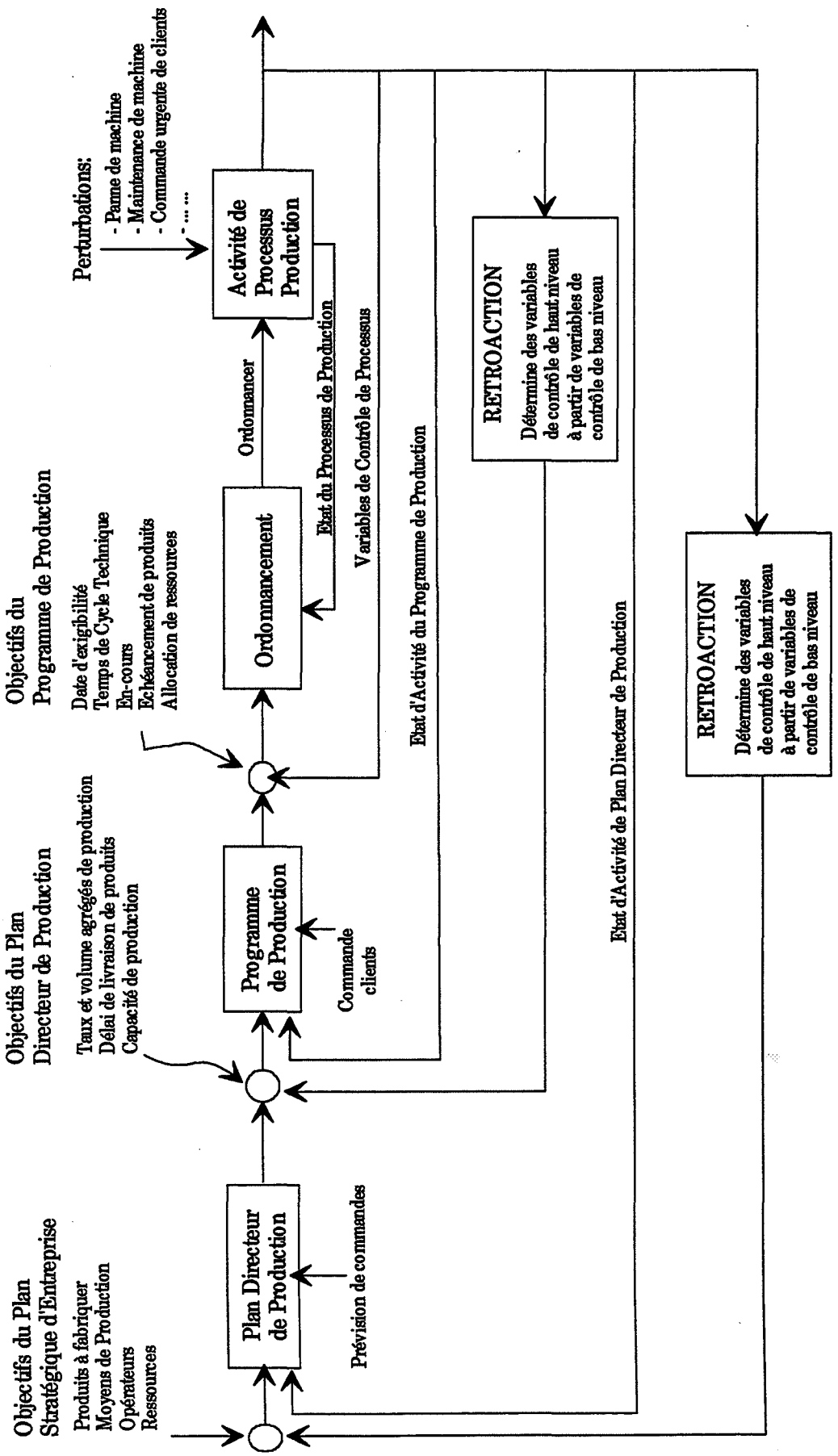


Figure 3-25 Interactions des Décisions du Système de Production

3) Un sous-ensemble d'entités décisionnelles de gestion/pilotage spécifiant les règles de gestion de processus du déroulement de la production (objets interactions, incidents ou spécifications).

Si l'avantage de l'approche à objets est de fournir une flexibilité, réutilisabilité et fiabilité aux systèmes à modéliser, nous devons introduire une démarche d'analyse pour la construction des modèles de l'application. Comme nous avons constaté dans le chapitre II que plusieurs approches d'analyse existent, nous utilisons la méthode SADT pour spécifier les étapes de la construction d'une application par l(approche à objets puisqu'elle propose une universalité des concepts et une simplicité d'utilisation.

La méthode SADT est souvent employée au préalable à la réalisation du schéma directeur, pour mieux connaître l'existant. Elle s'avère également intéressante pour dégager, à partir d'une analyse des besoins et des modèles du domaine (figure 2-4, chapitre II), les spécifications fonctionnelles du modèle de simulation à concevoir (figure 3-26). C'est dans cet aspect là que nous proposons notre démarche de la construction des modèles d'application: une démarche fonctionnelle du cycle auteur/lecteur pour mieux comprendre l'application à construire et ses principales fonctionnalités.

La méthodologie développée est centrée sur l'utilisation d'une technique: la simulation de processus par l'approche orientée-objets. Cette méthode se décompose en quatre grandes phases:

- la prise et l'analyse des données (analyse du domaine et analyse de l'application)
- la modélisation
- la simulation
- l'interprétation des résultats et conclusions de l'étude.

Les diagrammes SADT (actigramme) suivants décrivent l'enchaînement des activités de cette méthodologie (figure 3-26).

Après l'étude de modèles du domaine et l'étude de l'application, nous collectionnons les informations sur l'application étudiée, l'environnement, la recherche et les contraintes d'entreprise afin de produire des classes d'objets de l'application (physiques, décisionnels ou informationnels) (figure 3-27) et établir les modèles correspondants de cette application dans le but de la simulation (figure 3-28).

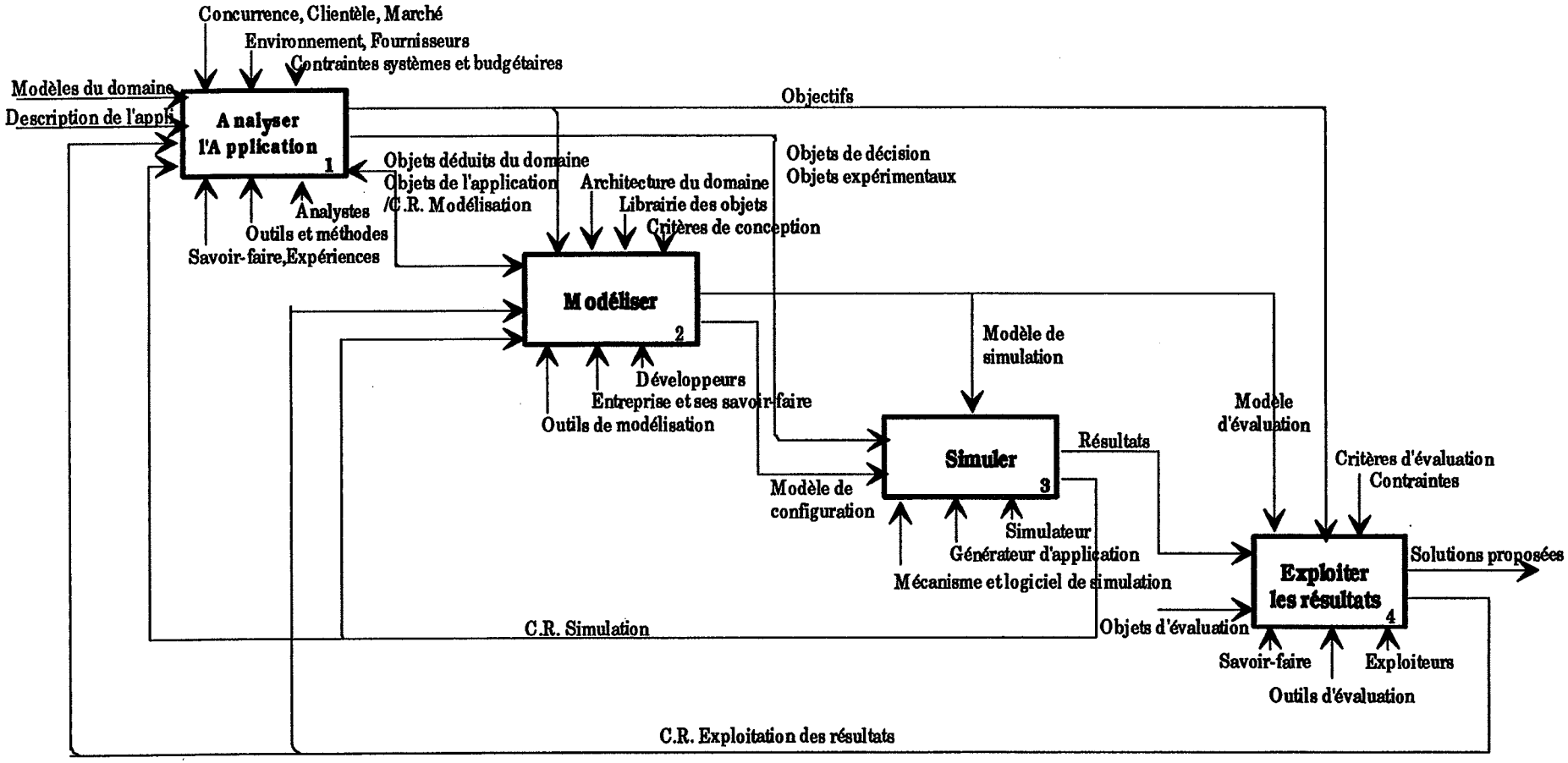


Figure 3-26 Schéma AO: Concevoir un Système de Simulation de Production

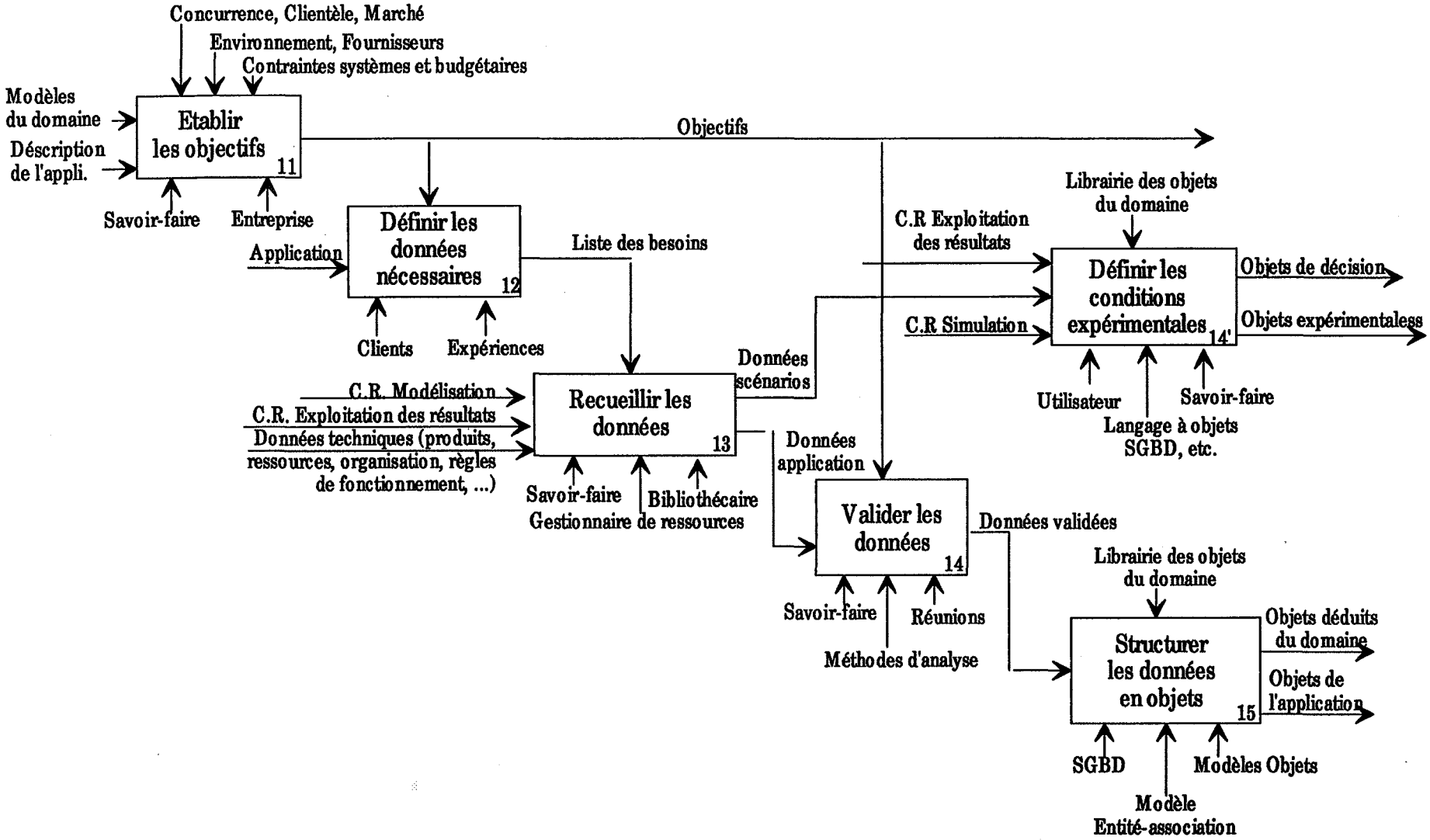


Figure 3-27 Diagramme A1: Analyser l'Application (les Systèmes de Production)

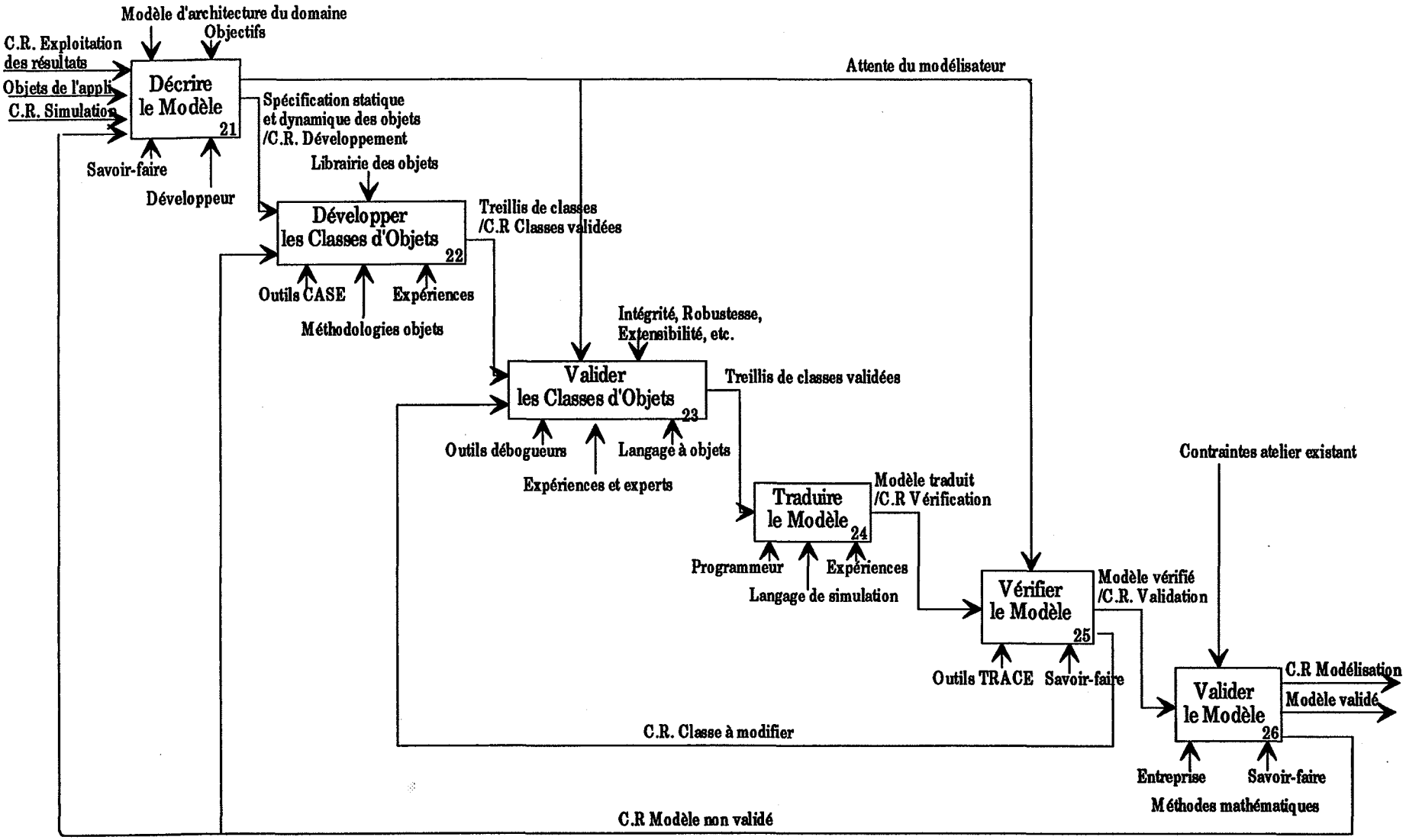


Figure 3-28 Diagramme A2: Modéliser l'Application (Un modèle de Système de Production

IV Conclusion

Dans ce chapitre, nous avons proposé une méthodologie d'analyse en nous inspirant des techniques de l'analyse du domaine et de l'analyse orientée-objets dans le but de construire un cahier des charges pour la simulation des systèmes de production.

Ce processus est subdivisé en deux étapes: l'analyse du domaine et l'analyse de l'application. Dans l'analyse du domaine, on établit le contexte principal dans lequel les informations utilisées dans le développement de logiciels sont identifiées, recueillies et organisées pour rendre ces informations réutilisables quand on met à jour ou quand on crée de nouveaux systèmes similaires. Le but de l'analyse est d'essayer d'aboutir à un consensus sur le vocabulaire et la théorie d'un domaine: un système de production est constitué de trois sous-systèmes (physiques, décisionnels et informatiques) dans lesquels trois populations interviennent (produits, ressources et moyens de production) et ces trois populations seront identifiées à travers cinq types d'objets (physiques, rôles, incidents, interactions et spécifications).

L'analyse de l'application prend les modèles de l'espace de problème, créés pendant la phase d'analyse du domaine, comme des entrées et se concentre sur l'application courante à construire afin de transformer les modèles du domaine en modèles d'application. Ces modèles sont représentés par trois modèles d'objets d'application: modèles informationnels qui décrivent les caractéristiques ou les attributs d'objets et leurs relations entre les classes d'objets, modèles des transitions d'état qui décrivent le comportement temporel d'objets et modèles de communication qui fournit une récapitulation graphique de la dynamique ou des interactions entre les objets (les échanges d'informations internes et avec le monde extérieur).

Les résultats de ces deux étapes d'analyse sont illustrés dans un schéma A1 du modèle SADT (figure 3-26) dans le but de fournir des classes d'objets pour la construction des modèles de simulation et des scénarios de simulation. Le chapitre suivant est consacré à la mise en ouvre de ces classes d'objets pour les systèmes de production.

Chapitre IV Conception d'un Modèle de Simulation des Systèmes de Production par l'Approche Objet

I Introduction

La conception concerne la spécification et la modélisation des éléments de la solution de problèmes du domaine ([BOOCH 92], [COAD et al. 91]). Concevoir un système consiste donc à transformer la représentation de problèmes en une représentation de résolutions. De nombreuses méthodes ou techniques de conception existent. Certaines d'entre elles en utilisation sont de type:

- procédural (conception structurée),
- orienté-données (data driven),
- logique,
- orienté-accès (une technique pour la conception d'interface utilisateur),
- orienté-objets,
- déclaratif,
- etc.

Chaque méthode a ses admirateurs et ses détracteurs. L'approche orientée-objets est une combinaison des techniques procédurales et orientées-données. Cette approche constitue un processus de conception qui repose sur un principe d'organisation du logiciel autour des objets relatifs à l'application que l'on souhaîte traiter, plutôt que sur les fonctions que l'on souhaite voir réalisées (c'est en s'appuyant sur les composants les plus stables du logiciel, ceux qui sont les moins susceptibles de changer, que l'on peut obtenir la souplesse et la réutilisabilité recherchées). Dans le chapitre précédent, nous avons analysé les systèmes de production avec l'approche à objets. Dans la phase de l'analyse, nous essayons de modéliser l'espace du domaine ou le sujet du discours qui est une perception fonctionnelle et partielle du système réel. Le but de l'analyse est de produire un cahier des charges qui spécifie la fonctionnalité du système. En utilisant le modèle objet, ce cahier des charges contient des classes d'objets essentiels du domaine, leurs structures et les services qu'ils doivent fournir et les relations entre ces classes.

Dans la phase de conception, nous devons tout d'abord détailler, pour chaque objet identifié par l'analyse, son comportement et ses intercommunications avec d'autres objets (messages) afin de satisfaire les problématiques posées par le cahier des charges de l'application. Ensuite, nous réexaminerons les classes d'objets du domaine: raffiner, étendre ou réorganiser ces classes

afin d'améliorer la réutilisabilité et la modularité, et de profiter ainsi au maximum des concepts d'héritage et de liaison dynamique. Enfin, nous devons ajouter d'autres objets additionnels exigés par la résolution logicielle de problèmes tels que les objets d'interfaces, les objets d'application, les objets utilitaires ou mathématiques, etc. La figure suivante (figure 4-1) décrit la relation générale qui existe entre l'analyse et la conception.

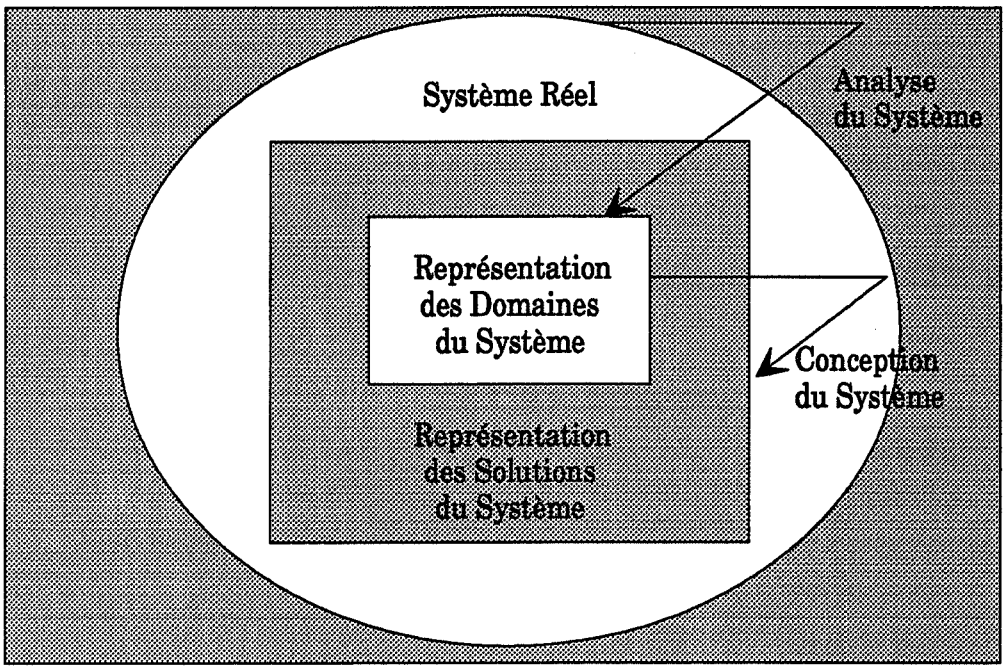


Figure 4-1 Relation entre l'Analyse et la Conception

La conception par objet (CO) consiste donc d'une part, à réexaminer les comportements et les communications des classes d'objets sémantiques (conception de haut niveau ou conception de l'application) qui représentent des entités physiques ou des concepts décrivant le système et son comportement afin d'améliorer la réutilisabilité, la modularité, etc.; d'autre part à définir des classes d'objets (conception de bas niveau ou conception des classes d'objets) qui reflètent la structure interne du logiciel tels que la hiérarchie des classes sémantiques, les objets d'interface, les objets d'application et les objets auxiliaires/utilitaires [MONARCHI et al. 92] (figure 4-2).

Tout objet du monde réel possède une activité (comportement) autonome. Les objets communiquent entre eux et avec l'extérieur par des messages. Or les méthodes traditionnelles à objets comme la méthode BOOCH [BOOCH 92], la méthode YOURDON ([COAD et al. 90], [COAD et al. 91]) et la méthode MEYER [MEYER 88], la méthode SHLAER/MELLOR [SHLAER et al. 92], la méthode OMT [RUMBAUGH et al. 91], fournissent des moyens pour identifier et définir des objets séquentiels, c'est-à-dire, des objets qui fournissent une encapsulation d'organisation (structure) et certains services prêts à être utilisés. On dit souvent que ce sont des objets séquentiels ou des objets passifs [COMMUNI 93].

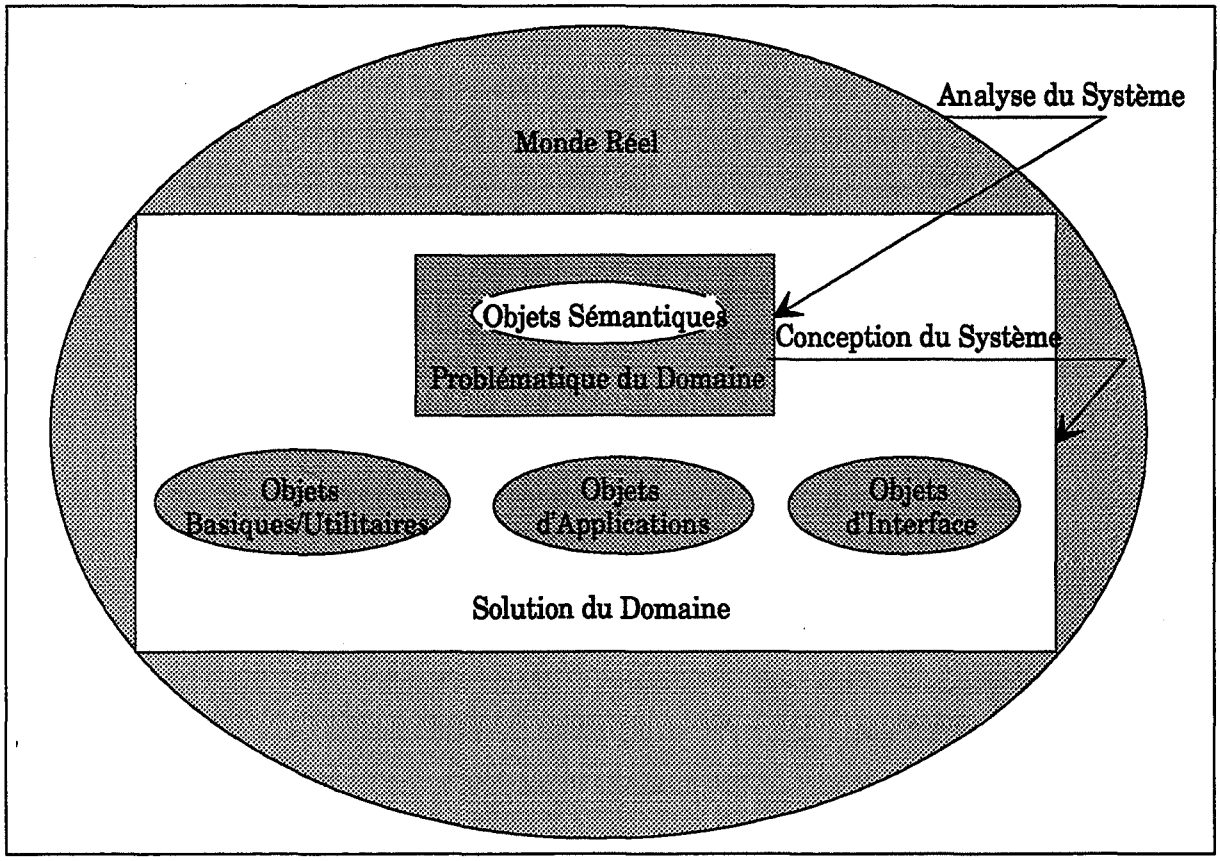


Figure 4-2 Analyse et Conception Orientées-Objets

Dans un modèle de simulation basé sur la notion de processus (chapitre II), un monde réel est modélisé par deux types d'objets: des objets *actifs* et des objets *passifs*. Les objets actifs ont un comportement autonome (on appelle aussi des objets contrôles) dont l'évolution dans le temps est définie par ses processus. Dans ce chapitre, nous considérons les moyens de production, les procédures de commande, etc., comme des objets actifs, et les produits, les pièces, les messages qui circulent dans les systèmes de production comme des objets passifs (objets circulants, objets de transactions). Bien sûr, nous pouvons faire l'inverse: définir les processus autour des produits et considérer les moyens de production et de transport comme des ressources pour des opérations effectuées sur les produits. C'est un choix sur le coût (effort) de modélisation et les indicateurs de performance sortis de la simulation projetée [BENSLEY et al. 92]. Nous pensons que les moyens de production sont plus stables par rapport aux produits qu'ils fabriquent, et surtout que notre objectif de simulation est de rendre compte des lois représentatives du fonctionnement des systèmes de production dans sa globalité. Après la conception du comportement des objets actifs et la conceptualisation des processus de production, nous construisons, dans ce chapitre, la hiérarchie des classes d'objet sémantiques pour la simulation des flux et la hiérarchie des classes d'objets pour la résolution du problème.

La distinction entre la phase d'analyse et de conception par objet est ambiguë et imprécise ([CRIBSS et al. 92], [HENDERSON et al. 90]). Si on examine la définition des classes d'objets sémantiques, l'analyse orientée-objets concerne plutôt la définition de la structure de classes d'objets et l'identification des relations entre ces classes, et la conception orientée-objets concerne la définition du comportement et la hiérarchisation des classes d'objets. Il n'existe pas, lors d'une analyse ou d'une conception par objet, d'une approche purement descendante ou purement ascendante; en fait il s'agit d'effectuer des "allers-retours", de manière à améliorer et à raffiner progressivement le modèle conceptuel du logiciel d'application et la hiérarchie des classes d'objets du domaine ([BOOCH 93], [HILL 93], [CASTELLANI 93]).

II Conception du Comportement des Classes d'Objets

Les méthodes traditionnelles de l'analyse et de la conception par objet définissent des classes d'objets séquentielles [CAROME 93]. Ceci est la conséquence du modèle de communication entre objets: lors du lancement du programme, un seul objet est actif, entre le moment où un message est émis et où le résultat est reçu, l'émetteur reste bloqué, et un objet n'est actif que pendant le calcul de la réponse. Ainsi, à un instant donné, un seul message est actif.

Dans le monde réel, les systèmes de production sont faits de multiples objets ou groupes d'objets situés à divers niveaux, communiquant entre eux et avec l'extérieur, dans les deux sens, explicitement ou implicitement. Idéalement, il faudrait que chaque objet repère sa production (entrée/sortie) et ses échanges de messages (informations) avec d'autres objets (ou avec lui-même) d'une part, entre l'organisation et l'environnement du système d'autre part, afin d'établir un comportement plus proche de la réalité dans l'entreprise.

Pour qu'un objet puisse fonctionner plus ou moins indépendamment des autres, il doit avoir un contexte ou un environnement partiel vis-à-vis des autres et être muni d'un contrôle interne qui coordonne les méthodes qui lui sont associées. Cette notion nous oblige à munir les objets d'une activité autonome ou d'un *processus de contrôle d'activité*. On parle d'objets actifs (contrôles), c'est-à-dire, des objets qui ont une activité autonome et peuvent travailler indépendamment et concurrentiellement avec d'autres objets et qui sont capables de:

- prendre ou ignorer les messages extérieurs,
- comprendre les messages et découvrir des possibilités de connaissances (ou précisément des réactions aux messages),
- traiter les messages conformément à son comportement aux normes fixés,
- arrêter ou reprendre leur activité ou l'activité d'un autre objet actif,

- obtenir des soutiens (aides) d'objets décisionnels (par le biais d'appel d'une méthode d'un objet décisionnel) ou des informations d'environnement (informations de contexte) dans le but de prendre des décisions (on dit qu'il communique avec l'extérieur),
- etc.

Dans le but de réduire la complexité de modélisation et des algorithmes de simulation, nous simplifions les objets du monde réel en deux grandes catégories: des objets passifs et des objets actifs (appelés aussi **"agents"** en intelligence artificielle). L'objet actif est différent de l'objet passif par ses primitives de création et de manipulation des processus. Un objet autonome est un objet actif qui possède un flot local de contrôle d'activités (une méthode spéciale appelée dorénavant *exécuter ou fonctionner*) que l'on appelle *un script de processus* qui définit la séquence d'exécution des méthodes d'un objet actif. En d'autres termes, les objets actifs ont des primitives de création de processus et des mécanismes de communication et synchronisation entre eux pour qu'ils puissent posséder un comportement autonome.

II.1 Définition du Script de Processus (Comportement) d'Objet Actif

Le schéma suivant (figure 4-3) représente une organisation générale d'un objet actif et son comportement. Il y a deux modes de communication entre un objet actif et son environnement:

- **Communication synchrone ou directe**: il existe un lien direct avec l'objet qui émet le message destiné à cet objet actif. Dans le cas d'une machine, par exemple, quand un transporteur arrive dans le stock amont vide, il va provoquer un message de démarrage de l'activité de la machine. On dit aussi que l'objet actif reçoit un message direct. Cela peut être un message de changement de mode de fonctionnement, par exemple un message comme "Si la machine N° 5 tombe en panne, alors la machine N° 3 doit cesser de fabriquer le type de pièce 4; la machine N° 3 peut changer le mode de sélection de pièce dans son stock amont ou remettre en cause tout ses heuristiques de fonctionnement". Ces messages sont générés par des objets actif qui ont un lien explicite avec cet objet.

- **Communication asynchrone ou indirecte**: l'objet actif a un tampon qui reçoit les messages extérieurs. Quand un message arrive, il ne perturbe pas l'activité courante de l'objet. Par exemple, à chaque machine est associée un tampon pour mémoriser les articles (messages) qui lui sont destinés. La machine prend en compte ces articles quand elle est libre. En informatique, on parle de la communication producteur/consommateur ou boîte aux lettres, un mécanisme qui permet aux objets actifs de communiquer sans nécessiter la présence simultanée des interlocuteurs.

Dans ce schéma, nous distinguons dans l'activité d'un objet actif:

- la perception et l'acquisition: décodage du message,
- la cognition: reconnaissance et interprétation du message,
- la décision: mode de réaction au message,
- bloquer, démarrer ou redémarrer l'activité de l'objet et déléguer ou traiter le message reçu: réaction plus ou moins autonome de l'objet et/ou communication à d'autres objets.

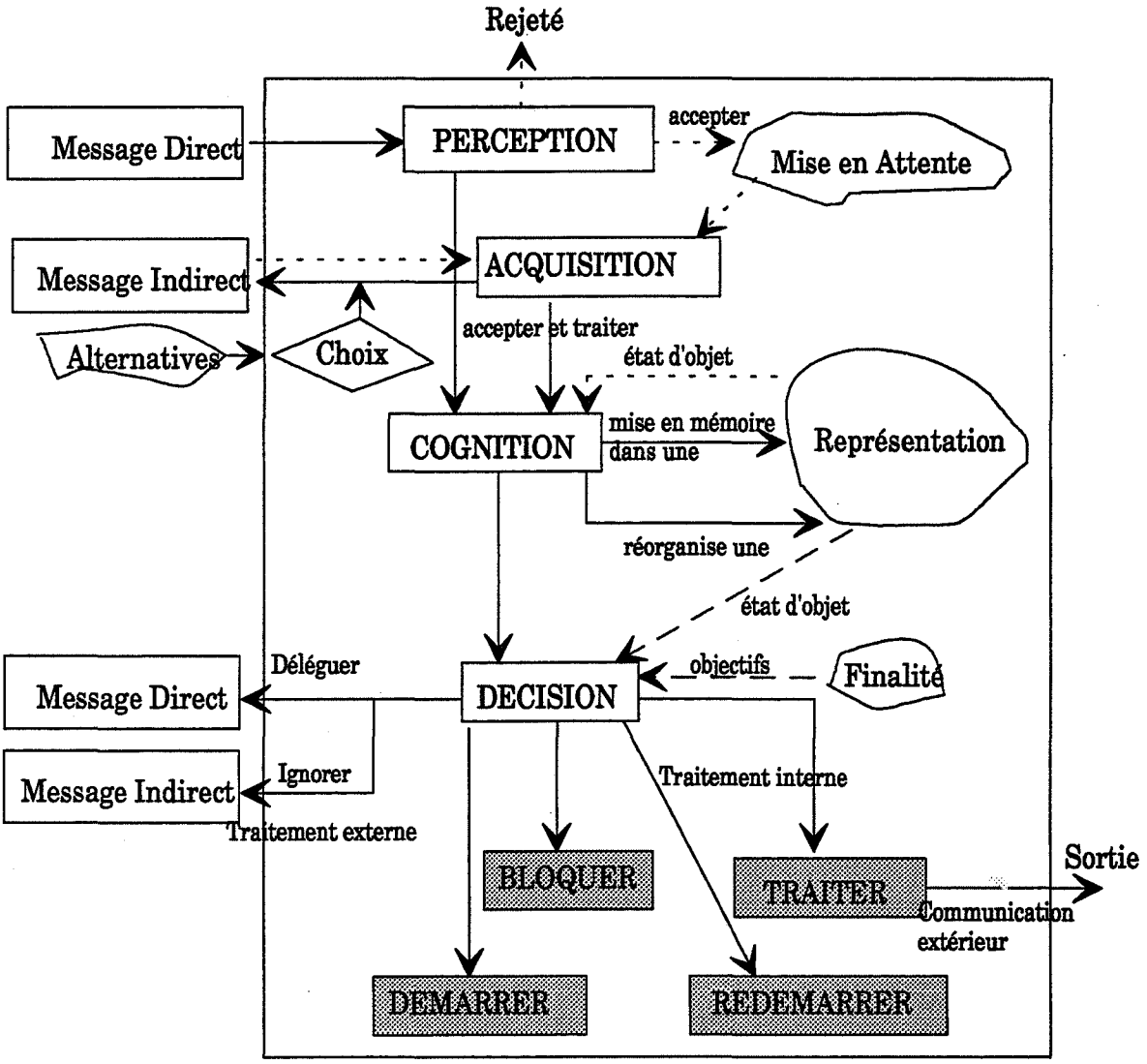


Figure 4-3 Script de Processus d'un Objet Actif (Concurrent)

II.1.1 Perception et Acquisition

Le message direct, parvenant à l'objet, sa perception peut être rejetée ou acceptée. Le rejet (qui sans doute n'est jamais total mais peut laisser une trace inconsciente) correspond à un filtrage de ce que l'objet n'est pas capable de percevoir ou de ce que l'objet ne désire pas percevoir ces

messages. Le message accepté peut être mis en attente lors que l'activité courante de l'objet ne peut pas être interrompue, ou être interprété immédiatement.

Les messages indirects, provenant de l'extérieur et stockés dans le tampon, et les messages directs mis en attente sont pris en compte à un temps donné par cet objet selon son comportement temporel. Le choix d'un message dépend des alternatives disponibles (règles de priorité pour des files d'attente, par exemple) et le mode de fonctionnement de cet objet (flux tirés ou flux poussés pour une machine, par exemple).

II.1.2 Cognition

Les relations entre perception ou acquisition et cognition sont très étroites. La cognition est la reconnaissance, la compréhension, l'interprétation du message compte tenu des connaissances et des mécanismes d'intelligence de l'objet. Pour une machine, ce pourrait être, par exemple, le traitement du message compte tenu des données (état de la machine) et des programmes stockés (méthodes de la machine) dans sa mémoire. L'arrivée d'une pièce dans le stock amont, par exemple, est un message indirect; après acquisition de la pièce, la machine va chercher l'opération à réaliser sur la pièce dont les caractéristiques de l'opération sont décrites dans sa gamme d'usinage. Les données instantanées et événementielles comme le temps d'usinage, les ressources associées sont mises à jour et les données historiques de la machine et de la pièce sont archivées, etc.

En d'autres termes, la cognition dépend de chaque objet en fonction de ses représentations (structure ou attributs) et de sa capacité à réorganiser ces représentations (méthodes associées) sous l'influence de nouveaux messages. Schématiquement, trois cas peuvent se présenter:

- le message met à jour et mémorise l'état d'un ou de plusieurs objets (dans une représentation),
- le message modifie ou réorganise la structure d'un ou de plusieurs objets,
- enfin le message peut déclencher une prise de décision interne de l'objets dans le but de lancer une action interne ou externe.

Dans les deux premiers cas, les messages déclenchent immédiatement l'exécution d'une méthode séquentielle (qu'on appelle aussi procédure ou méthode normale). Ces méthodes correspondent aux concepts d'objets traditionnels. Dans le dernier cas, les messages peuvent provoquer un changement du comportement spatial ou temporel, c'est-à-dire, qu'ils vont déclencher une méthode concurrente (ou un processus en termes de la simulation qui provoque un avancement du temps du système).

II.1.3 Décision

La décision consiste à choisir, à déclencher, à arrêter ou à redémarrer un traitement externe ou interne selon la signification du message reçu, l'état et la finalité de l'objet.

L'objet peut déléguer le message à un autre objet ou l'ignorer s'il n'est pas capable de le traiter. Il peut aussi faire appel aux méthodes d'autres objets par l'envoi d'un message direct. Par exemple, une machine très surchargée peut déléguer un ordre de fabrication (pas au niveau de l'article) qui lui est destiné à une machine remplaçante si c'est nécessaire. Cet objet assure la transmission du message reçu, élaboré et codifié vers d'autres objets.

L'objet peut déclencher un traitement interne ou plus précisément une réaction interne au message: il va

- se comporter différemment dans sa discussion (une exécution des méthodes) avec son état courant et les objectifs déduits de sa finalité (sous quelle forme);
- modifier consciemment ou non une décision, une manière de faire, toutes choses inexplicables pour quelqu'un d'autre extérieur qui ne serait pas au courant de la circulation du message (encapsulation des méthodes décisionnelles d'objets): déclencher, arrêter ou interrompre une action.

II.1.4 Action

Une action se passe dans un *contexte*. Le contexte décrit l'ensemble des informations permettant à une action de savoir où il en est dans l'exécution de son script de processus (de son activité). Un processus est une unité atomique d'exécution des activités, qui peut être virtuellement un processeur ([BUHR et al. 90], [DORSEUIL et al. 91]). Avant de savoir où il en est dans l'exécution de son activité, l'objet va consulter le contexte de son processus.

Un processus a aussi des états: il peut être *élu*, *éligible*, *bloqué* et *terminé*. Quand on passe de l'exécution d'un processus à celle d'un autre processus, il y a une *commutation (changement) de contexte*: une opération qui consiste à remplacer le contexte d'un processus utilisant le processeur par celui d'un autre processus qui doit devenir élu suite à l'arrivée d'une action issue de la décision (voir chapitre V). Dans la vie courante, c'est comme si nous posons une question sur un autre sujet.

Les décisions prises par un objet peuvent conduire aux actions suivantes:

- bloquer un processus inconditionnellement,

- bloquer un processus en attendant qu'un ou plusieurs autres processus soit (soient) terminé (s),
- débloquer un processus,
- interrompre un processus,
- démarrer ou arrêter un processus,
- etc.

Ces actions causent un changement de contexte. Différentes implémentations [SCHIPER 86] sont possibles: *moniteur* en Concurrent Pascal, *tâche* et *rendez-vous* en ADA, *coroutine* en Modula II et en C++. Un objet actif et autonome possède au minimum un script de processus ou un *thread* en terme informatique qui décrit ce qu'un objet doit faire dans sa vie. Ce thread a aussi une notion de "séquentialité" [DAVID 93] dans ce terme qui rappelle le fil de la vie d'un objet actif. Nous conceptualisons ici les processus des objets au niveau de la simulation des flux de production. La description sur l'implémentation sera présentée dans le chapitre V.

II.2 Conceptualisation des Processus de Production

II.2.1 Opération (Tâche)

Une opération est une action d'un objet actif. En terme informatique, c'est une action (tâche) ou une activité du processus. Un système en temps réel est composé, en général, de plusieurs tâches qui s'exécutent en parallèle. Une tâche peut être:

- *activée* (élue, éligible ou bloquée): elle démarre lorsque la tâche était arrêtée;
- *élue*: elle monopolise la ressource processeur pour une durée t;
- *bloquée*: elle suspend la tâche appelante pour une durée indéterminée.

Il est à noter que la quasi-totalité des opérations industrielles (figure 4-4 [BARBIER 89]) peuvent être formalisées à partir d'un jeu réduit d'opérations élémentaires en termes de modification d'états d'objets: transformation, assemblage, transport, etc.

II.2.2 Processus

Un processus est une logique d'enchaînement d'opérations (tâches). Pour notre part, nous définissons un processus par rapport à un ou plusieurs objets actifs. Un processus détermine indissociablement un ensemble d'opérations exécutées par un objet actif. Il est à noter qu'un processus détermine des contraintes d'antériorité de certaines opérations sur certaines autres et non pas, sauf cas particuliers d'un processus purement séquentiel, une chronologie (thread) d'enchaînement d'opérations.

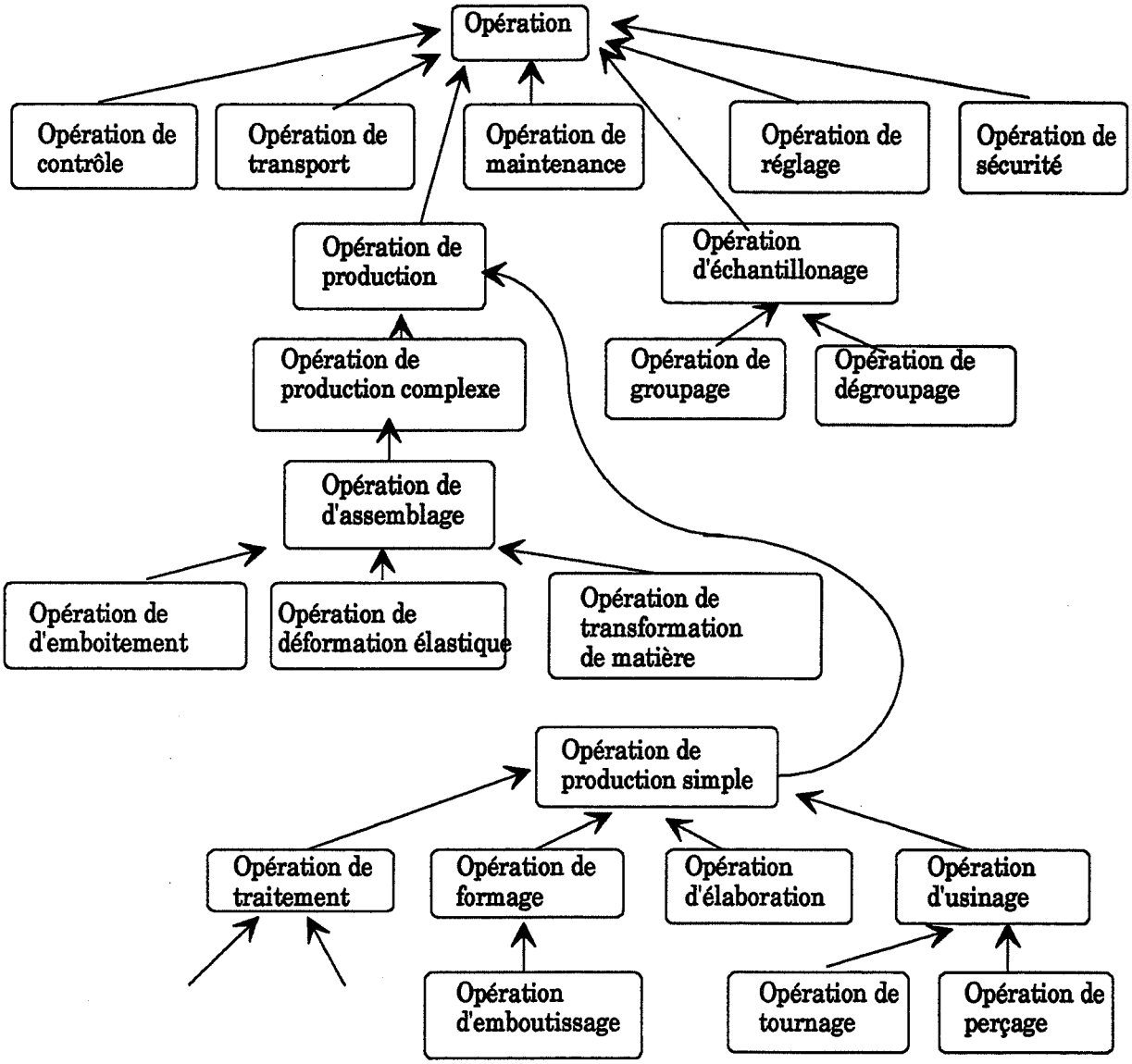


Figure 4-4 Une Classification d'Opérations dans les Processus de Production

En fait, un processus répond lui-même à la définition d'une opération dans la mesure où, considéré globalement, il réalise la genèse d'objets en aval (sorties du processus) à partir d'objets en amont (entrées du processus).

Un processus spécifie donc la décomposition d'une opération "*agrégée*" en opérations à un autre niveau respectant une logique d'enchaînement déterminée. Cette décomposition est récursive et peut être poursuivie jusqu'au niveau des opérations élémentaires de la figure 4-4. Barakat [BARAKAT 91] a proposé 4 niveaux pour l'abstraction des opérations et des processus: *action*, *acte*, *opération* et *fonction*. Pour notre part pour la simulation des flux de production, nous travaillons sur le niveau acte et opération. Dans la partie suivante, les

attributs et les méthodes pour la construction de modèles d'atelier sont identifiés et structurés selon ces deux niveaux.

III Construction de Modèles de Simulation

III.1 Architecture de Modèles

Un système de production est un système ouvert sur de nombreux environnements évolutifs avec lesquels il est en interaction permanente. Les changements de l'environnement induisent des réactions du système ou le conduisent à des actions anticipatrices. Celles-ci sont réalisées, dans un modèle de simulation, par un ensemble de schémas de comportement et de règles de conduite (un ensemble de scénarios en terme de simulation) définis dans le module de pré-simulation (figure 4-5).

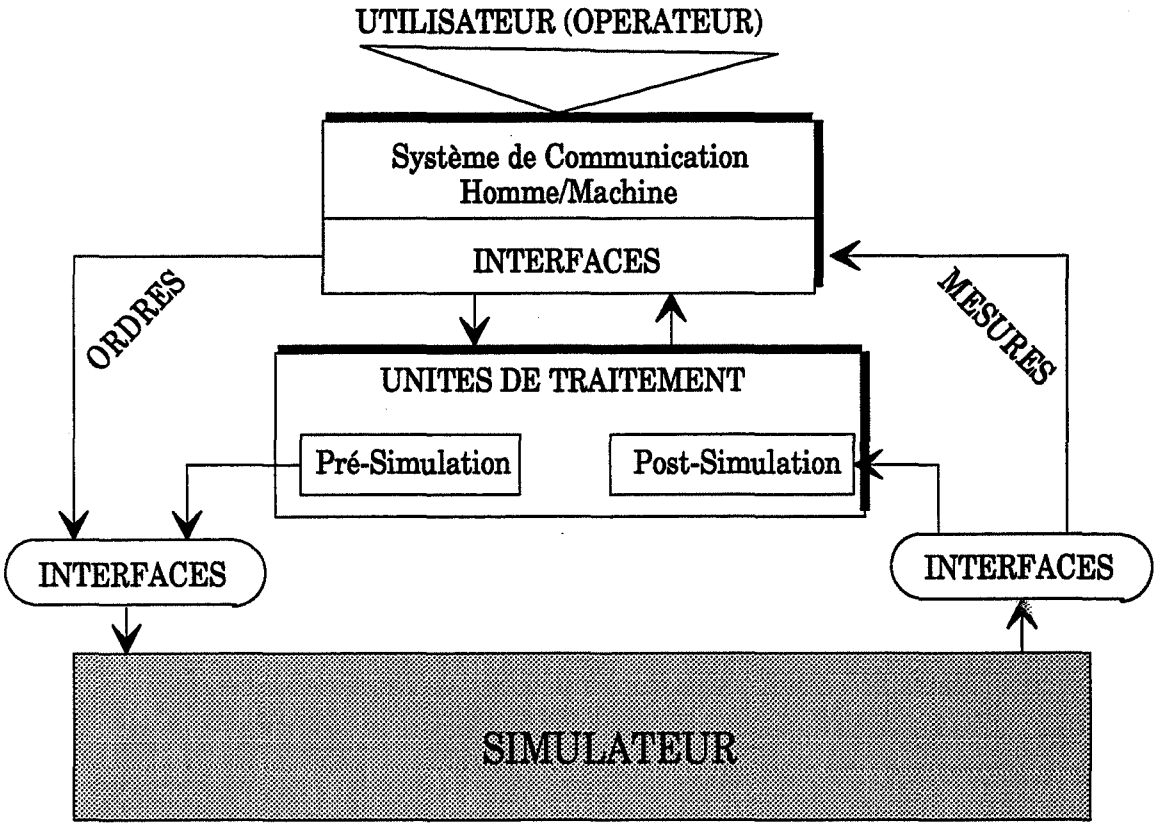


Figure 4-5 Architecture des Domaines d'Application en Vue de la Simulation

Un module de *pré-simulation* comprend les fonctions configuration, planification et ordonnancement, modélisé essentiellement par des objets d'interfaces sémantiques qui représentent les attributs structurels et contingents des objets sémantiques tels que les articles,

les ressources et les moyens de production, des algorithmes qui traduisent la logique M.R.P. ou des règles de pilotage. Les résultats de la simulation sont traités dans un module appelé *post-simulation* qui a pour objectif l'analyse et l'évaluation des performances de tel ou tel scénario défini dans le module de pré-simulation. Ce module est aussi modélisé par des objets d'interfaces qui représentent essentiellement les attributs historiques et statistiques des objets sémantiques (comme histogramme, diagramme de gant, etc.) en incluant des algorithmes d'analyse et de statistique (comme calcul des variances, des moments, des moyennes, etc.). Ces deux modules constituent l'unité de traitement d'interface homme-machine qui peuvent fonctionner indépendamment du module de simulation (on parle de simulation hors de ligne).

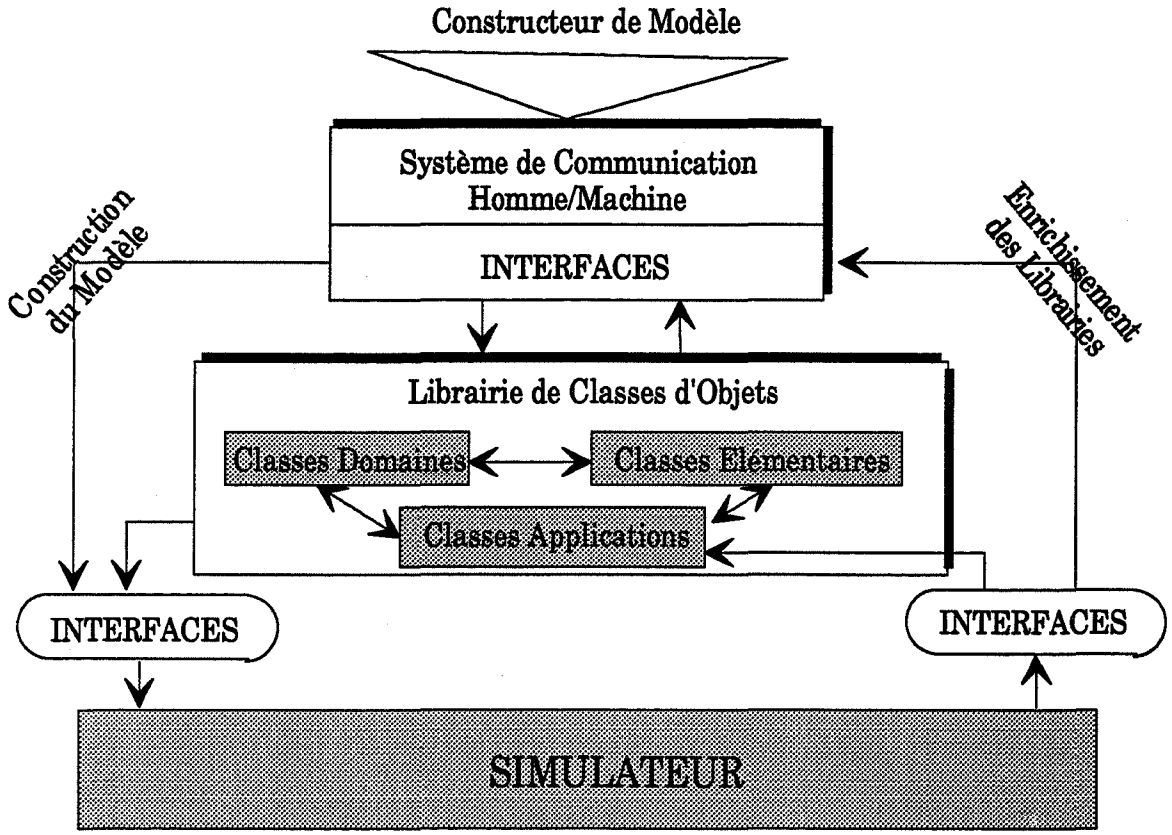


Figure 4-6 Architecture de Construction de Modèles de Simulation

Un module de simulation (simulateur) produit certains aspects du fonctionnement dynamique d'un système de production afin de répondre à des interrogations sur son fonctionnement définies dans le module de pré-simulation. Notons que la simulation ne permet pas de trouver de façon directe des solutions à des problèmes de production (comme les logiciels d'ordonnancement ou comme les systèmes experts). C'est pourquoi les modules de pré- et surtout de post-simulation constituent une partie très importante du modèle de simulation. La figure (figure 4-6) montre l'intervention des utilisateurs sur la reconfiguration de scénarios et de modèles de simulation à l'aide des classes d'objets dans la librairie et la mise à jour de la

librairie des classes. Nous détaillons ici seulement la modélisation du système de production, c'est-à-dire le module de simulation que nous allons construire.

III.2 Modélisation de Production

Notre objectif est de représenter, selon des degrés de finesse différents, le déroulement de la production. Une telle modélisation doit être structurée, intégrée et informatisable.

- *Structurée*, pour avoir plusieurs niveaux de détail dans la description du processus de fabrication. En conception, on cherche à impliquer progressivement l'équipement de production au fur et à mesure des choix technologiques. Inversement, le suivi de la fabrication conduit à ignorer les actions technologiques de l'équipement pour n'en retenir que leurs effets fonctionnels sur le produit. Pour notre part, nous travaillons sur le niveau action sous l'aspect fabrication [BARAKAT 91].

- *Intégrée*, par une description explicite de chaque opération non seulement en termes d'effets (conséquences) sur le produit (méthodes associée au produit) mais aussi en termes de fonctionnement des équipements (agrégation des opérations au niveau processus ou définition des méthodes associées à différents niveaux des moyens de production).

- *Informatisable*, en vue d'élaborer des outils interactifs de simulation d'ateliers. Néanmoins, les systèmes de production ne sont pas toujours entièrement informatisables. L'intervention d'opérateurs humains joue fréquemment un rôle essentiel: soit qu'elle permette des solutions moins onéreuses et plus efficaces, soit qu'elle se révèle indispensable, dans le cas par exemple de la mise en oeuvre de règles de gestion non formalisables, de l'exploitation d'un certain savoir-faire ou du maintien d'un certain niveau de vigilance.

Nous considérons qu'un atelier est constitué de n machines qui communiquent par le biais de m transporteurs. Les produits qu'un atelier peut produire sont modifiables et variables. Les activités de production se situent en trois niveaux: opération, processus (machine, station, transporteur, procédure de pilotage) et atelier (figure 4-7).

- *Niveau Opérationnel.* On identifie à ce niveau les différentes actions des objets actifs. Quatre types d'opérations peuvent être analysées vis-à-vis des objets actifs et de leur environnement: transformation, transport, assemblage, démarrage (déblocage)/arrêt (blocage). La transformation peut être physique ou logique avec ou sans équipement

associé, etc. L'opération de transport ou d'assemblage peut être vue comme une transformation spéciale.

- *Niveau Processus.* On définit à ce niveau, autour des objets actifs, la séquence d'opérations (macro-opération) qui sera représentée par un processus.

- *Niveau Atelier.* On définit des règles de pilotage ou l'interface qui contrôlent la circulation des objets passifs qui seront transformés ou transportés par différentes machines ou transporteurs dans l'atelier. Le comportement d'atelier est caractérisé par ceux des machines et des transporteurs.

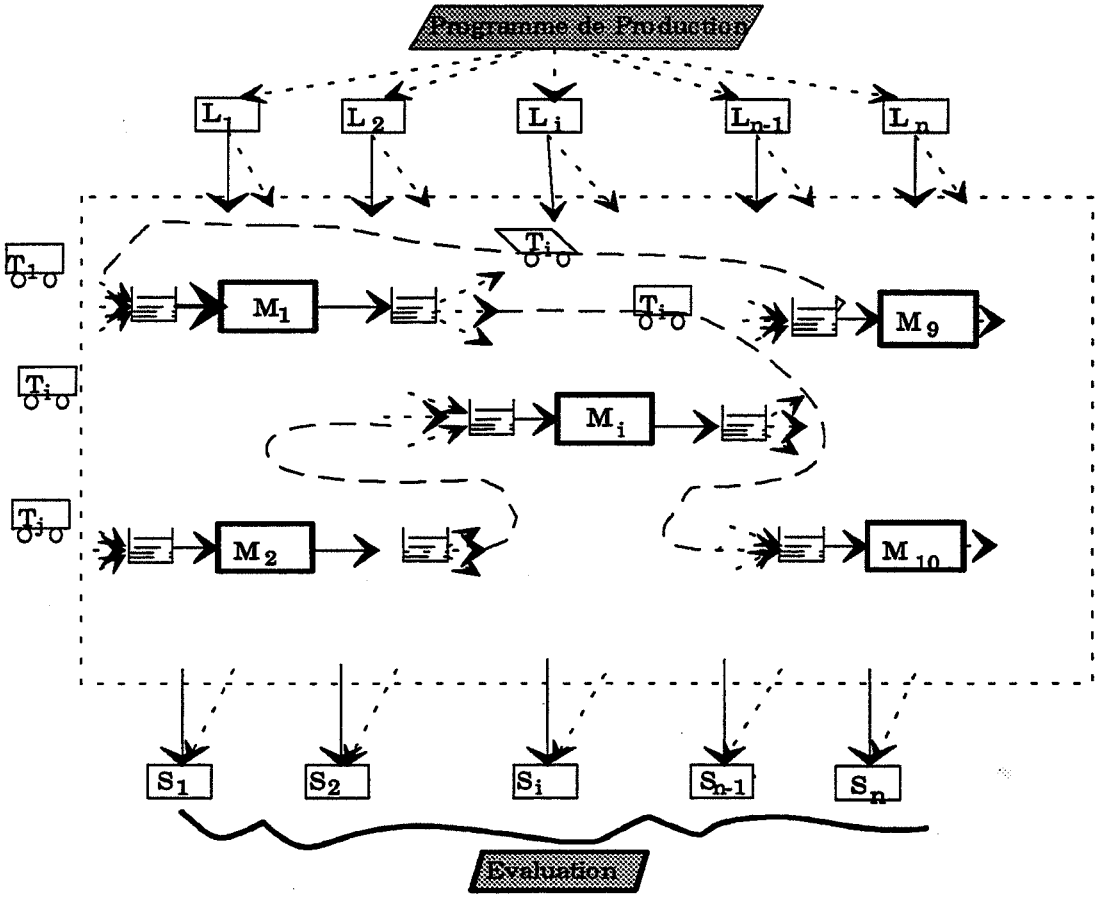


Figure 4-7 Modèle d'Atelier

La production ne se passe que lorsqu'il y a des produits à fabriquer dans l'atelier. Les trois niveaux des activités de production formalisées ci-dessus sont spécifiés par rapport aux moyens de production dans le but de définir des décisions locales qui sont indépendantes de produits à transformer. En fait, la modélisation de production doit inclure non seulement les moyens de production, mais également les produits que l'atelier peut fabriquer et les règles globales de

conduite d'atelier. Nous proposons une structure globale de modèle de simulation qui contient quatre niveaux de modélisation (figure 4-8).

- *Niveau Ressource*. On définit les typologies des ressources (machines, transporteurs, outils, opérations) et leurs comportements élémentaires sans considérer les caractéristiques des produits à fabriquer. En d'autres termes, il s'agit de définir des classes d'objets de moyens de production indépendamment de leurs utilisations contextuelles.

- *Niveau Atelier*. On intègre à ce niveau l'organisation de la production. Autrement dit, on définit la structure ou l'organisation des moyens de production (flow-shop, job-shop, etc.) et intègre les gammes de produits à fabriquer qui influent les fonctionnement des moyens de production dans cet atelier.

- *Niveau Unité de Production*. Puisque la structure de l'atelier et les produits à fabriquer sont définis, on définit à ce niveau des règles locales pour piloter la circulation des flux matières (de produits) entre les moyens de production telles que l'affectation des ressources, la sélection d'une opération, etc.

- *Niveau Système de Production*. C'est le niveau le plus élevé dans le modèle de simulation. On définit ici les règles globales de gestion qui sont en général déduites d'un système de gestion assistée par ordinateur telles que les règles de lancement d'un ordre de fabrication, l'investissement des moyens de la production, la réorganisation de la production, ou l'adoption d'une autre politique de gestion (passe de la méthode M.R.P. à la méthode juste-à-temps, par exemple).

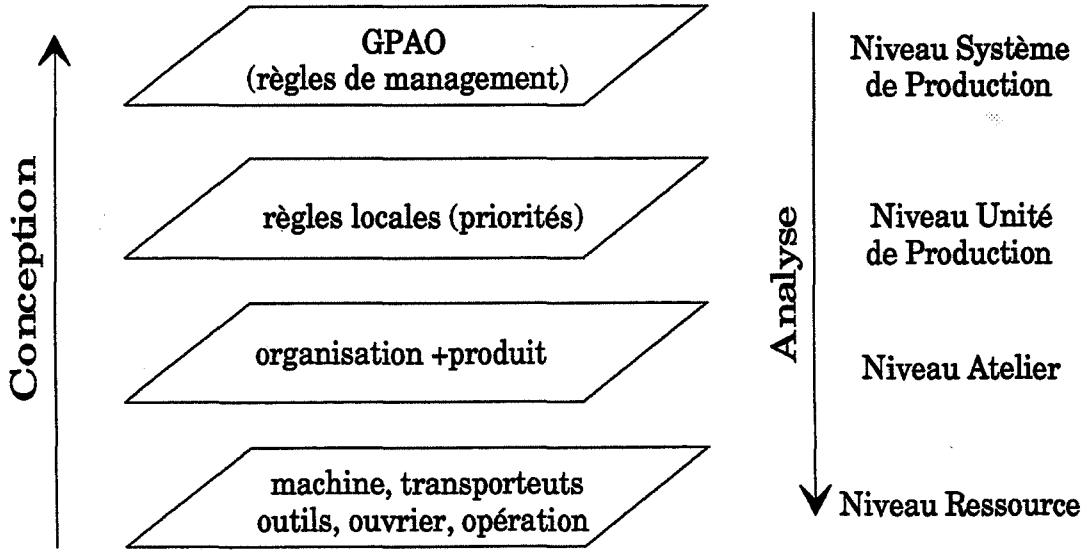


Figure 4-8 Niveaux de Modélisation dans la Simulation

III.3 Intégration des Décisions dans le Modèle de Simulation

Nous avons défini qu'une décision, au niveau d'un objet actif, est de choisir, de déclencher ou d'arrêter une action (une méthode d'objet). Au niveau du système, le système de décisions consiste à enchaîner un ensemble d'actions simples ou complexes conformément aux objectifs fixés. Comme nous l'avons constaté, la simulation ne permet pas de trouver de façon directe des solutions à des problèmes de production, nous devons intégrer les décisions, dans les simulations traditionnelles, dans les scénarios correspondants (avant l'exécution du modèle); puisque la modification (reconfiguration) du modèle au cours de la simulation est difficile voire impossible. Dans l'approche orientée-objets, les mécanismes de liaison dynamique et le polymorphisme nous permettent d'intégrer les décisions au moment de l'exécution du modèle. Nous pouvons donc intégrer des décisions dans le modèle de simulation de trois façons en fonction des objectifs souhaités:

- *La surveillance*: la fonction de décision n'est pas automatisée. Il ne s'agit que d'analyse de données, de visualisation de ces données (sous formes graphiques ou textuelles) et d'indication d'alarmes à travers différents objets d'interface au cours de la simulation (on les qualifie de décisions interactives). L'opérateur (utilisateur) du modèle prend des décisions selon les contextes et son expérience. Les informations ne circulent que dans un seul sens: du modèle de simulation au système de communication homme/machine (figure 4-5). L'information sortie de la simulation est matérialisée normalement par les fichiers de scénarios (ou une base de données).

- *Le mode guide opérateur*: ce mode complète en fait le mode précédent en apportant des traitements un peu plus élaborés. Le système ne se contente plus de transmettre les sorties de simulation pour qu'elles soient facilement interprétables par l'opérateur mais il prend des décisions (dans les modules pré et/ou post simulation des unités de traitement) et propose à l'opérateur des actions possibles que celui-ci n'a plus qu'à appliquer.

- *Le contrôle automatisé*: il s'agit de la véritable règle de pilotage automatisée de la simulation, c'est-à-dire d'une structure en boucle fermée. L'opérateur n'intervient plus dans les activités de la simulation. Ceci est fait à travers différents algorithmes ou règles de pilotages intégrés dans le module de simulation.

Nous nous intéressons ici à la structuration et à la définition de cette dernière sorte de décision. Les deux premières façons qui seront en général prises en compte par la simulation sont définies après la construction du modèle de simulation ou en dehors du module de simulation (simulation hors-ligne), puisqu'elles dépendent du modèle de production à simuler.

IV Construction des Hiérarchies des Classes d'Objets Sémantiques

Plusieurs concepts importants doivent être présentés avant d'aborder les hiérarchies des classes d'objets pour la simulation des systèmes de production. Ce sont les perspectives, les points de vue, les relations et le cycle de vie des classes d'objets, etc.

IV.1 Trois Perspectives de la Représentation par Objets

Du point de vue de la modélisation pour la simulation, il est préférable de regarder les classes d'objets sous trois perspectives: perspective élémentaire, perspective orientée-modèle et perspective orientée-application ([MCLAREN et al. 88],[YE et al. 92a]) (figure 4-9).

- La *perspective élémentaire* concerne la construction des classes d'objets de base que contient le système, ainsi que les mécanismes pour atteindre la flexibilité et la maintenabilité qui sont essentiels pour le support de la simulation. Dans cette perspective, deux choix importants doivent être considérés: choix d'une approche de simulation qui adapte à l'architecture de simulation déterminée pendant la phase d'analyse du domaine (figure 3-3, par exemple) et choix d'un langage de programmation à objets pour implémenter des objets de base pour la simulation. La simulation discrète est basée sur la notion d'événement ou de processus. Les événements ou processus sont mis dans une liste appelée *échéancier* qui est un noyau de synchronisation du système (*scheduler*). Dans l'approche par objet basée sur la notion de processus, le système à simuler est décomposé en un ensemble de processus concurrents. Chaque processus représente une entité qui évolue dans le temps par opposition aux autres entités. Le "scheduler" contrôle l'exécution des processus et permet de les interrompre ou de les reprendre. Cette perspective sera détaillée dans le chapitre V.

- Dans la *perspective orientée-modèle*, l'utilisabilité et la compréhensibilité des classes d'objets sont primordiales, elles doivent donc bien couvrir le domaine à modéliser. Il s'agit en effet d'élaborer un ensemble minimum de classes d'objets (objets de transactions, objets de moyens de production, objets de décision, etc.) permettant de représenter tout type de systèmes de production. L'objectif est de fournir une librairie de classes d'objets exploitables pour la construction des modèles de simulation afin d'en tester la validité et d'en évaluer les performances. Dans cette perspective, nous ne précisons pas qu'un objet est actif ou passif. Tous les objets sont considérés actifs et autonomes: on les appelle classes d'objets contrôles qui encapsulent non seulement les informations, mais également les comportements temporels ou dynamiques (interactions entre les objets).

- La *perspective orientée-application* concerne essentiellement l'enrichissement et l'instanciation des classes d'objets du domaine, la construction et l'implémentation de nouvelles classes de l'application et la construction des modèles de diverses applications particulières (en précisant les rôles d'objets (actifs ou passifs) et en connectant ces instances des classes pour une application) conformément au cahier des charges des applications.

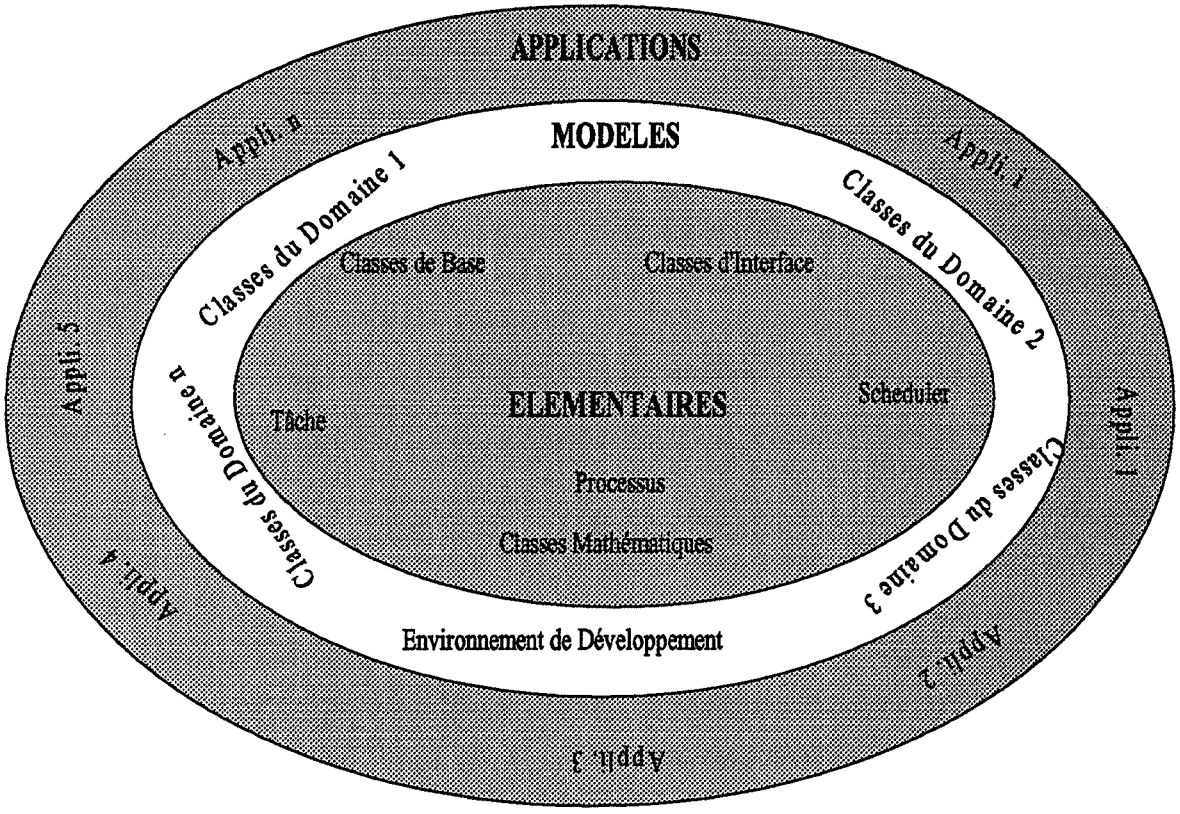


Figure 4-9 Perspectives des Classes d'Objets

IV.2 Les Points de Vue des Objets (Versions d'Objets)

La perception que l'on peut avoir d'un objet varie dans le temps, d'un individu à l'autre suivant le niveau d'observation (contexte). Un objet machine vu par différents acteurs tels les concepteurs, les contrôleurs, les gammistes ou les opérateurs est différent. Dans les systèmes à objets, ces différentes perceptions correspondent à autant de *points de vue* (*perspectives* en intelligence artificielle, *vues* en base de données, *représentation* en conception assistée par ordinateur) différents d'un même objet. Au sein d'un même point de vue, la connaissance (méthodes associées) détenue sur un objet peut *évoluer*, en particulier elle peut être affinée ou changée dans sa vie (on parle de *version* d'objets).

Les points de vue et les versions d'objets sont orthogonales. Pour un même objet on doit pouvoir prendre en compte un nouveau point de vue et/ou en affiner d'autres. Plusieurs possibilités existent dans les systèmes à objets. Ces concepts sont pris en compte par trois mécanismes.

- Une modélisation explicite ou implicite des *points de vue*. Une modélisation explicite consiste à enrichir les concepts d'objets (sous différents points de vue) et à intégrer ces concepts dans un même objet. Une machine dans le modèle de simulation, par exemple, peut être vue sous des points de vue organisationnels, processus, flux, agrégation/désagrégation, niveaux de décision [YE 93], points de vue qui permettent d'identifier ses structures et les relations associées. Une modélisation implicite (on ne rajoute pas de nouveaux concepts) consiste à utiliser des concepts existants, par exemple l'agrégation, la spécialisation multiple; et ces points de vue cohabitent au sein de la classe agrégée en terme de composition ou de la classe parent en terme d'héritage.

- Un mécanisme de *classification*. La classification consiste à rechercher à quelle classe une instance (un objet) particulière peut être rattachée [RECHENMANN 88]. Si, par exemple, la connaissance sur une instance est enrichie, le mécanisme de classification tente de la rattacher à une classe plus spécialisée que celle de sa création (polymorphisme et liaison dynamique). De manière plus générale, la classification permet de prendre en compte l'évolution des connaissances détenues sur un objet.

- Un mécanisme de *multi-instanciation*. La multi-instantiation consiste à permettre le rattachement simultané d'un objet à plusieurs classes qui ne sont pas directement ou indirectement des spécialisations les unes des autres ([RIEU et al. 91], [NGUYEN et al. 92]). Elle autorise donc le rattachement d'une instance à des classes comportant des sémantiques différentes de celle drainée par sa classe de création. Il ne s'agit plus d'affinage ou d'une remise en cause d'un point de vue de l'objet mais bien d'un enrichissement de connaissances sémantiquement différentes, c'est-à-dire d'une prise en compte d'un nouveau point de vue sur un objet.

Bien que la classification et la multi-instanciation présentent certains avantages, nous nous intéressons plutôt à la modélisation explicite et implicite des points de vue par différentes relations: spécialisation/généralisation, agrégation/désagrégation, association, etc.

IV.3 Les Relations des Classes D'Objets

Au niveau de l'analyse, nous avons défini qu'une relation est un lien stable entre deux objets différents, qui permet à tout moment, connaissant l'un d'entre eux, de connaître l'autre. Dans les systèmes à objets, il faut distinguer les relations au niveau de l'application et au niveau des classes d'objets. Ces deux niveaux fournissent une division naturelle de la conception de haut niveau et de la conception détaillée (de bas niveau). Les relations au niveau de l'application ont pour but d'aider à modéliser l'espace du problème tandis que les relations au niveau des classes ont pour but de supporter l'implémentation des classes d'objets individuelles.

IV.3.1 Relations au Niveau de l'Application

Dans le processus de développement orienté-objets, la phase d'analyse et de conception de haut niveau consiste à modéliser l'environnement et la structure du problème. Les classes d'objets identifiées et développées à ce niveau correspondent aux entités ou concepts du monde réel. Les relations entre ces classes sont aussi identifiables dans le monde réel. Ces classes et relations sont nommées en général avec les termes du domaine à modéliser.

Bien qu'il existe un nombre infini de relations possibles dans l'espace de domaine, trois types de relations peuvent couvrir l'ensemble des relations: spécialisation/généralisation, agrégation et association. Ces trois relations, décrites au niveau de l'application, sont très similaires à celles au niveau des classes. La différence est dans l'utilisation de la relation et non pas dans leur définition. Une relation au niveau de l'application est une partie de modélisation du domaine, tandis qu'une relation au niveau des classes est orientée vers l'implémentation des classes.

- *Spécialisation/Généralisation.* Cette relation a pour but de développer un modèle hiérarchisé qui part des concepts abstraits pour définir, dans les couches successives inférieures, des entités qui sont de plus en plus spécifiques. La relation spécialisation est souvent désignée sous le nom d'un *lien est-un* ou d'un *lien sorte-de* (*is-a* ou *a-kind-of* en anglais). Elle définit une relation de pré-ordre (un ordre partiel) et interdit la possibilité des cycles dans la structure de spécialisation. Bien évidemment, une entité peut être une spécialisation de plusieurs concepts généraux (plusieurs points de vue, par exemple). A l'inverse de la spécialisation qui permet decrire des objets spécifiques à partir d'un autre objet, la généralisation a pour but de déduire des objets plus généraux, voire marginaux (objets abstraits), à partir d'un ensemble d'objets qui ont des caractéristiques communes. Cette relation est intimement liée au mécanisme de classification.

- *Agrégation.* Cette relation, très utile pour modéliser des systèmes importants, exprime le fait qu'une entité peut être décrite à partir d'autres entités: une agrégation est une union

d'objets formant un objet. Cette relation est souvent désignée sous le nom d'un *lien partie-de* (*is-part-of* en anglais). Tout système, et par conséquent tout objet, peut être décomposé en sous-systèmes, eux même décomposables en sous-systèmes et ainsi de suite jusqu'à l'obtention des sous-systèmes ou des objets primitifs. Cette modélisation d'un tout en parties de plus en plus fines est communément appelée *hiérarchie de parties* qui décrit les liens structurels entre les parties (objets). On trouve, dans un objet agrégé (composé), deux types d'information: les informations à chacune des parties et les informations propres au tout (qui résultent précisément de la composition de plusieurs objets au départ indépendants). Dans la hiérarchie de parties qui constitue l'objet agrégé, ces informations doivent être stockées à des niveaux logiques donc à des niveaux différents. Dans l'idéal, le tout doit connaître les parties qui le composent mais ces parties n'ont pas de connaissance du tout.

- *Association*. Cette relation permet d'établir un rapport entre deux entités dans lequel l'une d'entre elles détient des instances de l'autre. Cette relation a été largement abordée dans le développement des librairies à objets par des classes "conteneurs" ou "collections". Ces classes définissent des protocoles communs pour recevoir, enregistrer ou retourner des instances qu'elles possèdent; mais elles ne modifient pas ces instances (elles passent seulement des messages à ces instances). Dans un modèle de simulation, cette relation sert à modéliser différentes files d'attente, stocks, etc. L'association est différente de la relation agrégation par le fait qu'une association est un lien entre une classe et les "pièces" qui profitent des services de la classe "conteneur"; par contre une agrégation est une relation structurelle entre une classe et les parties de sa représentation.

IV.3.2 Relations au Niveau des Classes d'Objets

Les relations au niveau des classes surgissent quand nous conceptualisons l'implémentation d'une classe. Souvent l'implémentation d'une classe dépend des instances des autres classes. Les relations au niveau des classes peuvent être l'implémentation des relations au niveau de l'application ou l'implémentation des attributs relations entre les classes. Nous présentons trois relations importantes: message, composition et héritage.

- *Message*. La plupart des relations au niveau de l'application (agrégation et association) sont réalisées dans le modèle objet par l'envoi des messages entre les instances de deux classes impliquées. Un message est donc une requête adressée à un objet demandant l'exécution d'une opération, c'est-à-dire, une spécification d'une opération à exécuter sur un objet. Le message doit donc correspondre à une méthode spécifiée dans l'interface publique d'une classe dont l'instance sera le récepteur (un sélecteur) du message. Une méthode est une procédure (ou fonction) implémentant la réponse aux messages destinés à l'objet.

Un message peut également passer des informations à un objet destinataire sous forme d'arguments. Le lien entre une classe et les classes dont les instances sont figurées dans le message est appelé une relation *refers-to* (une *référence* dynamique). Le graphe produit par des envois des messages constitue le coeur de la structure d'un système à objets dans lequel l'envoi des messages est le seul moyen de communication et de synchronisation.

- *Composition.* Cette relation est une notion d'implémentation des relations qui correspondent aux relations agrégations au niveau de l'application. L'encapsulation de la classe est réalisée par la création des instances d'une ou plusieurs classes. Quand une classe est assemblée à partir des instances des autres classes, ces instances ne sont plus des objets indépendants. L'utilisateur peut ou non agir directement sur ces instances dépendant de l'implémentation de la classe. Si la classe est une *boite noire* dont les instances sont inaccessibles, l'implémentation s'apparente dans ce cas à la notion d'encapsulation. L'inconvénient de cette démarche est qu'elle oblige à définir l'ensemble du protocole à ce niveau et qu'elle écrase totalement la hiérarchie de parties définie au niveau de l'application. Si la classe est vue comme un ensemble de composantes accessibles, il devient nécessaire de contrôler l'accès à ces composantes (instances). Le problème qui se pose, lorsqu'on veut modifier la classe, est de savoir si elle est composante d'une autre classe complexe. On voit ici que la relation inverse *partie-de* ne peut en pratique être ignorée.

- *Héritage*. L'héritage est un mécanisme destiné à l'économie des systèmes: la relation spécialisation/génération au niveau de l'application préconise la décomposition des systèmes et la minimisation de leurs parties communes; au niveau des classes, l'héritage permet les partages d'objets et autorise la définition d'une nouvelle classe à partir de la (ou des) définition(s) d'une classe (héritage simple) ou de plusieurs classes (héritage multiple). Il faut donc distinguer deux types d'utilisation de l'héritage: *l'héritage pour l'implémentation* et *l'héritage pour la spécialisation*. L'héritage pour la spécialisation est une implémentation des relations entre les classes spécifiées au niveau de l'application par des relations spécialisation/généralisation. L'héritage pour l'implémentation signifie que la représentation interne des classes existantes est utilisée pour fournir une partie de la représentation interne de la classe à définir. D'autres concepts importants liés à l'héritage comme les variables de classe, les variables d'instance, la valeur par défaut, le polymorphisme, l'héritage statique, l'héritage dynamique, la liaison dynamique (résolution tardive), etc., sont bien expliqués dans la littérature ([BOOCH 92], [COMMUNI 90], [COMMUNI 92], [MEYER 88], [COAD et al. 90], [COAD et al. 91], [MCGREGOR et al. 92], [SHLAER et al. 92], [RUMBAUGH et al 91]), nous n'abordons pas ici ces mécanismes.

IV.4 Le Cycle de Vie des Classes d'Objets

Pour que la réutilisation joue un rôle plus important dans le développement du logiciel, les classes (composants du programme) doivent avoir une vie indépendante de l'application dans laquelle elles sont originairement développées. Le développement des classes s'adresse à la conception et à l'implémentation des entités (objets), qui peut bien évidemment aider dans la résolution du problème courant, mais ces classes doivent également rester suffisamment générales pour qu'elles puissent être réutilisées dans le futur projet. Le développement d'une application peut être considéré comme la coordination des instances de ses classes afin de produire une solution à un problème spécifique ou le développement d'un produit particulier. La figure suivante (figure 4-10) présente un cycle de vie de classes qui peut être orthogonal par rapport au cycle de vie de l'application (analyse, conception, implémentation, test et maintenance): l'identification de classes fait partie du cycle de l'application, mais les étapes du développement des classes sont indépendantes du développement d'une application particulière. Cette indépendance a pour but de produire une classe qui est un modèle le plus complet possible d'un concept ou d'une entité à représenter et qui est celle qui sera le plus probablement réutilisée dans les autres applications.

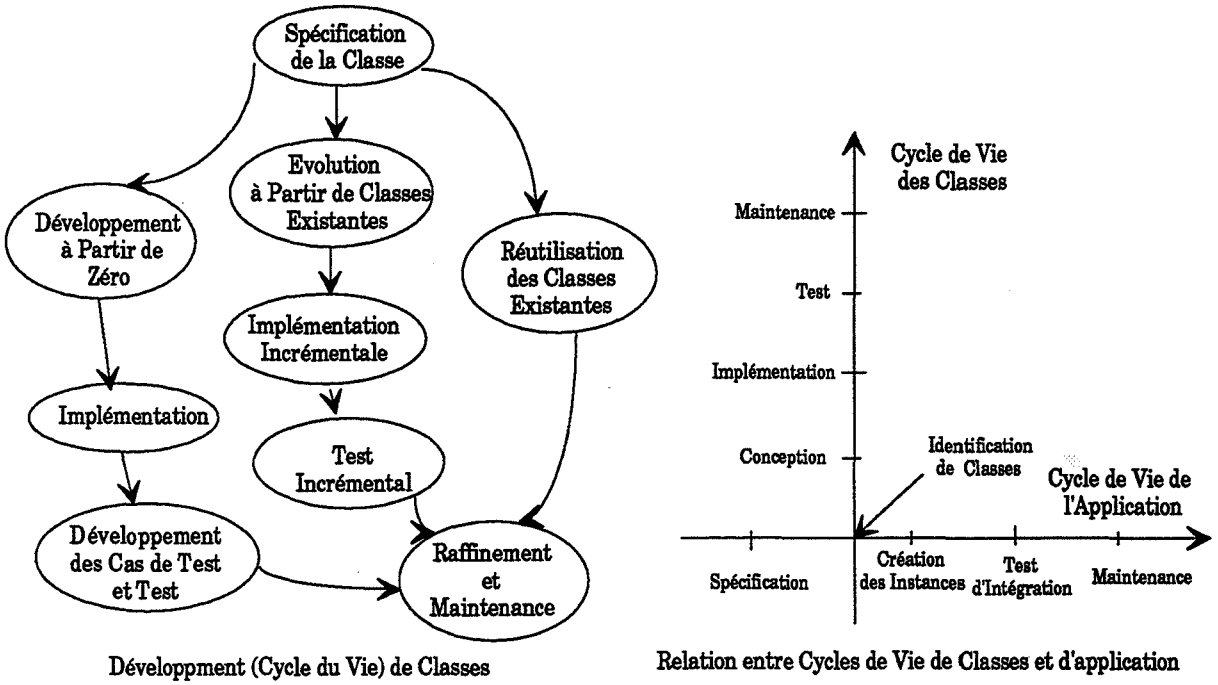


Figure 4-10 Cycle de Vie de Classes et Relations avec l'Application

Une classe identifiée dans l'application peut être développée de trois façons: développer une classe à partir de zéro s'il n'existe pas de classes similaires dans la librairie, réutiliser les classes existantes (c'est-à-dire créer des instances) ou faire évoluer les classes existantes (c'est-à-dire

spécialiser des classes) si nécessaire. Dans la plupart des outils pour la simulation orientée-objets, les classes essentielles pour la simulation (au point de vue de perspective élémentaire) sont définies comme dans les logiciels (dont nous disposons dans notre laboratoire du département): MODSIM II [BELANGER 90a], Meijin++ [NICOLAS 91], SIM Plus Plus [ROSE 92]. Le travail essentiel des développeurs est d'étendre ces outils de simulation vers le domaine de la production.

IV.5 Les Classes d'Objets dans la Simulation des Flux

Dans le chapitre III, nous avons identifié trois grandes catégories d'objets constituant un système de production: objets logiques ou transactionnels, objets moyens de production et objets décisionnels. Ces trois ensembles d'objets incluent tous les objets identifiés pendant la phase d'analyse du domaine: objets physiques, objets rôles, objets interactions, objets incidents et objets spécifications; et ils sont représentés dans un modèle de simulation par deux types d'objets: objets actifs et objets passifs. Le choix d'implémenter une entité du domaine (objet sémantique) comme un objet passif ou actif dépend du coût de modélisation et des indicateurs de performances sortis de la simulation. Un objet passif est aussi appelé objet circulant. Dans la partie qui suit, nous ne faisons aucune distinction entre un objet passif et un objet actif. Rappelons justement qu'un objet actif est un objet dérivé d'un objet *Process* dans lequel on peut définir un flot de contrôle local (script du processus) de l'objet.

IV.5.1 Définition des Classes d'Objets de Transactions

Le système de pilotage d'atelier reçoit des informations du niveau de décision hiérarchiquement supérieur. Les informations qu'il reçoit en amont sont des quantités de produits à fabriquer pour une date donnée. Un produit peut être une pièce simple ou assemblé à partir de plusieurs pièces (composants). Chaque produit est affecté à une classe **PRODUIT** qui est caractérisée par des attributs: nom de produit, quantité à fabriquer, date au plus tôt (date prévue de lancement), date au plus tard (délai de livraison), prix de vente, nomenclature, priorité (confiance aux clients), et des attributs caractérisant ses performances: coût de production, temps de production, temps d'avance/retard, etc. Les attributs sont essentiellement dérivés d'un système de GPAO et les méthodes associées sont issues principalement de la logique M.R.P.

Si un produit à fabriquer (ordre de production) est simple, il correspond à un ordre de fabrication (classe **LOT**), sinon, nous devons suivre sa nomenclature en utilisant la méthode M.R.P. pour calculer, pour chaque composant, sa quantité à fabriquer, la date au plus tôt et au plus tard, sa priorité, etc. On a donc besoin d'une classe **NOMENCLATURE** qui est une arborescence des composants pour faire le lien logique entre le produit et ses composants. Ces

nomenclatures définissent l'ordre pratique d'utilisation de différents composants et la hiérarchie d'exploitation.

Les composants ou pièces sont des entités réelles qui vont subir des transformations dans le système de production. Ils appartiennent à une classe **COMPOSANT (PIECE)** qui inclue non seulement les attributs (structurels et conjoncturels) induits de la classe produit (lien logique), mais également leurs propres attributs (instantanés, événementiels, historiques et statistiques). Leurs méthodes définissent les relations entre leurs attributs, les liens avec les produits à travers la nomenclature et les liens avec les moyens de production à travers la gamme de fabrication. La mise à jour des valeurs des attributs d'un composant est fait par les comportements des moyens de production. (Notons que nous n'avons représenté que, dans la figure 4-11, les attributs essentiels des classes d'objets concernées, les attributs détaillés, une pièce par exemple, peuvent référencer aux modèles informationnels d'objets (figure 3-23)).

La gamme de fabrication d'un composant (ou d'un produit) est l'ensemble des opérations que doit subir ce composant dans le système de production. La classe **GAMME** est donc une liste d'opérations qui décrit les tâches à effectuer séquentiellement sur les matières ou les composants pour aboutir à un produit fini. Les méthodes essentielles sont définies dans la classe abstraite supérieure (liste simple, liste double, ou tableau dynamique, etc.).

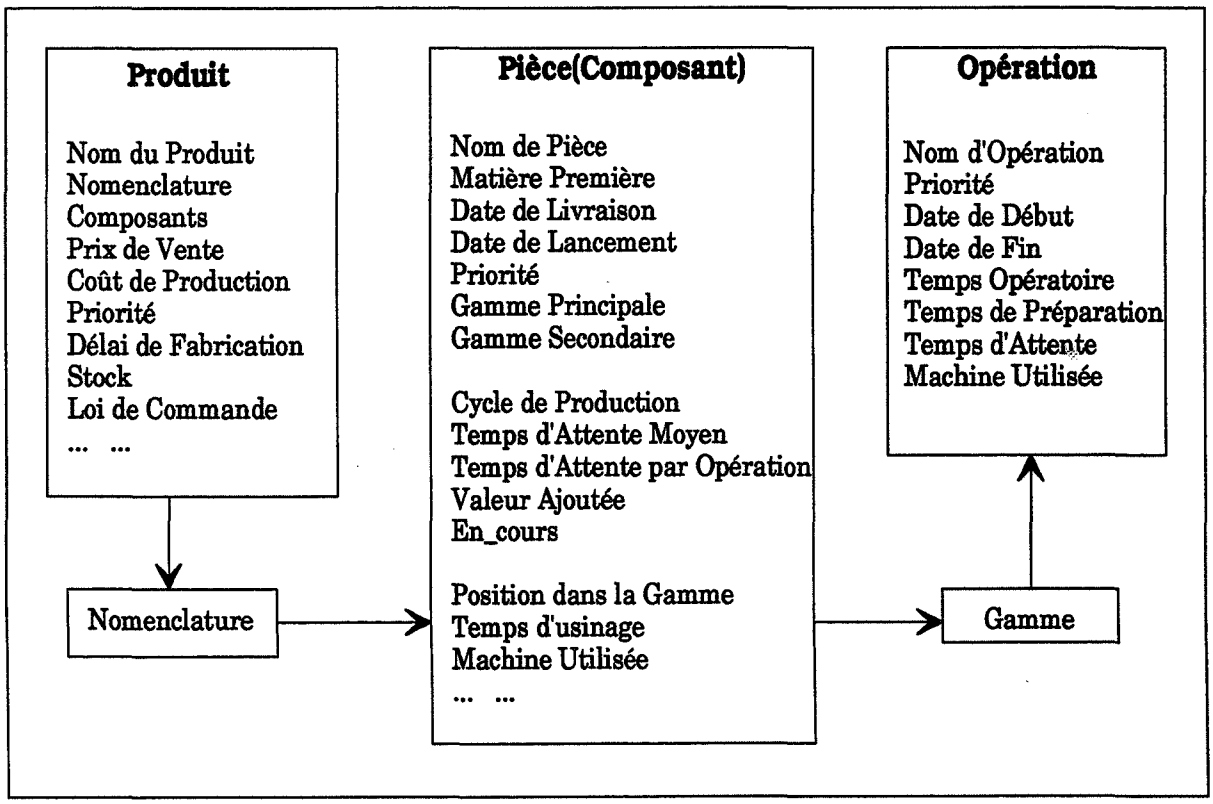


Figure 4-11 Liens entre Produit, Pièce et Opération

La classe **OPERATION** est la description d'une étape de fabrication sur une pièce qui contient comme attributs: le temps opératoire, le type de machine nécessaire, la priorité (par rapport au chemin critique), la date de début et de fin de l'opération, etc. La gamme de production est un ordre partiel (en d'autres termes, il peut exister dans une gamme un certain nombre d'opérations réalisables dans un ordre quelconque) mais la structure de représentation (tableau ou liste) est un ordre linéaire. Nous devons donc détailler l'objet opération pour qu'il permette de modéliser cet ordre partiel. Il s'agit en effet de raffiner l'objet **OPERATION** en **TRANSFORMATION**, **ASSEMBLAGE** et **TRANSPORT** (figure 4-12) afin d'inclure des attributs pour définir la relation de l'agrégation/désagrégation des composants (l'assemblage est une opération effectuée sur plusieurs composants différents en même temps; le transport est une opération qui agit sur un composant ou un lot (plusieurs composants)). Le mécanisme polymorphisme nous permet d'implémenter cette relation d'association.

Notons que les opérations constituant les gammes de fabrication sont affectées aux moyens de production de manière dynamique, c'est-à-dire que lorsqu'une opération est terminée sur une pièce, la consultation de la gamme va indiquer un couple opération/moyen à effectuer ensuite. Pour chacune de ces opérations, on explore le parc de moyens candidats (s'ils existent) à utiliser, en évaluant par exemple un critère de coût, ce qui permet de sélectionner un couple opération/moyen dynamiquement permettant de l'exécuter. Ceci est au coeur de la procédure de simulation; la recherche d'un équipement pour un ensemble de moyens candidats se fait en trois temps:

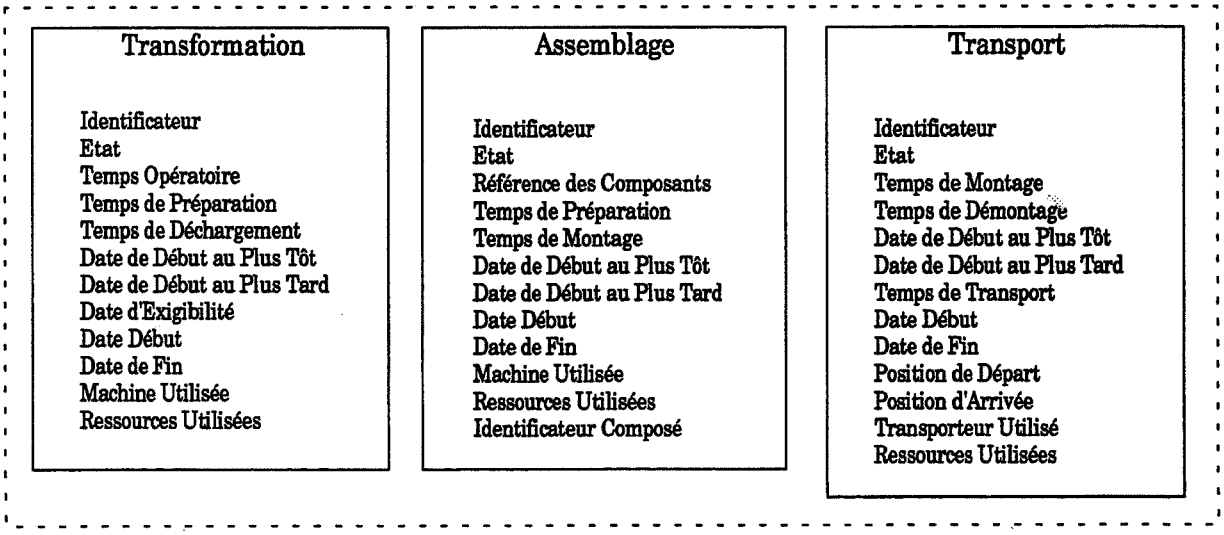


Figure 4-12 Structure des Objets Transformation, Assemblage et Transport

1) Détermination des machines susceptibles d'exécuter cette opération (par consultation des caractéristiques de l'opération concernante dans la gamme);

2) Pour chacune de ces machines:
- détermination des ressources de transport candidates pour acheminer la pièce de la machine actuelle vers la machine destination ainsi que de la disponibilité et du coût de ce transporteur;
- détermination du coût de l'opération sur la machine destination et de la capacité de cette machine.

3) Enfin une phase de choix, basée sur l'estimation des coûts, des délais ou des combinaisons des coûts et des délais, ce choix est alors traduit par différents algorithmes implémentés dans les files d'attente aval de la machine actuelle.

Cette procédure peut sembler complexe et onéreuse, cependant elle sera exploitée en regard de la structure du système de production et des gammes de fabrication à modéliser.

Les demandes exprimées par le niveau de décision hiérarchiquement supérieur donnent lieu au lancement de articles (pièces) dans l'atelier. La notion correspondant à cet ordre de fabrication dans le modèle de simulation est le concept lot. Les principaux attributs structurels et historiques de la classe **LOT** comprennent: contenu du lot (référence de composant), taille du lot lancé, taille du lot sorti, date de lancement, date de sortie, etc., et des attributs de performance: taux de rebuts, cadence de lancement, cadence de sortie, coefficient de perte, cumul des temps d'usinage/préparation, etc., qui sont aussi des méthodes prédéfinies par cette classe d'objet.

La notion de lot est une notion de groupement. Il peut servir comme unité de flux dans la simulation, c'est-à-dire dans le même état d'avancement. L'individualisation des pièces dans le lot nous permet d'observer plus finement les données (événements) de la production. Dans notre modèle de simulation, la notion de lot est utilisée comme une unité de lancement, il correspond éventuellement à un lot de transport (si le transporteur peut transporter plusieurs pièces en même temps). Les unités de base de la production est la notion de pièce (article). L'automatisation du découpage de lots ou groupement de pièces pour l'opération transport dépend des caractéristiques du transporteur et des pièces concernées. Ces fonctions implémentées dans les méthodes de l'objet transporteur seront précisées au moment de la construction du modèle de simulation par les utilisateurs.

Le modèle conceptuel de ces classes d'objets transactionnels peut être schématisé, en suivant un formalisme entité-association étendu, comme le montre dans la figure 4-13.

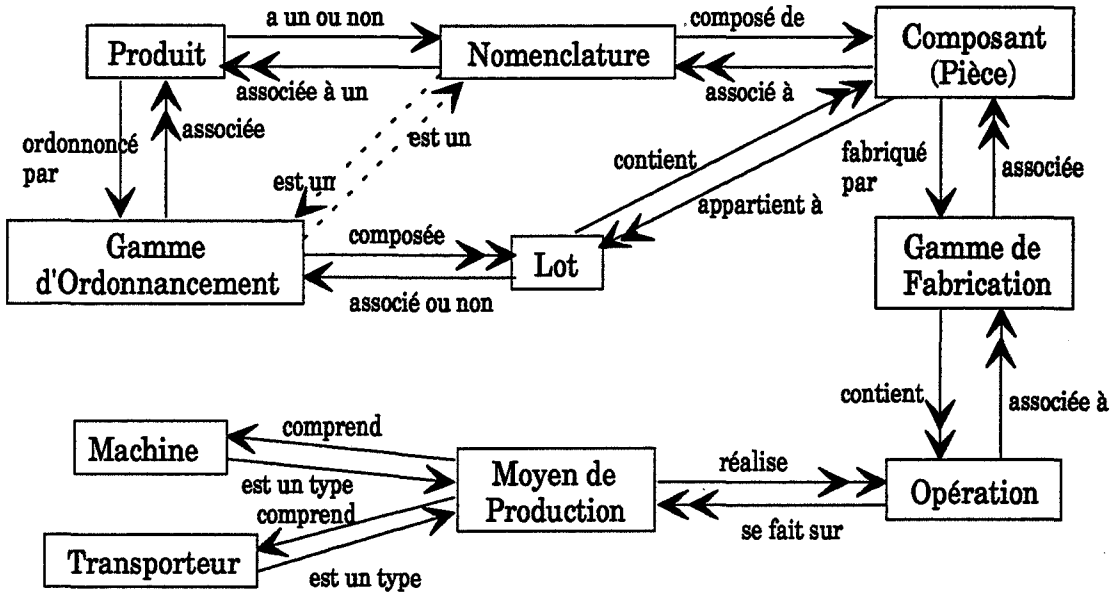


Figure 4-13 Modèle Conceptuel des Objets Transactions

Notons que la vision des moyens de production qui figurent dans le schéma précédent est considérée comme statique (ou une vision fonctionnelle), les points de vue dynamiques (comportements) sont détaillés dans leurs modèles de communication de processus (chapitre III) des objets correspondants. La gamme d'ordonnancement d'un produit peut être une instance de sa nomenclature ou une suite de phases d'ordonnancement qui contiennent une suite d'opérations à réaliser successivement sur une même cellule de gestion (unité de production) ou sur un lot (unité du flux) [YE 90]. Les gammes d'ordonnancement sont utilisées dans la planification à moyen terme et les gammes de fabrication servent à l'ordonnancement à court terme et à l'affectation des moyens de production à très court terme. Les liaisons entre le système de commande de la production et les gammes sont décrites dans la figure 4-14. La plupart de ces fonctions sont programmées par des unités de traitement dans le module de pré-simulation pour définir les scénarios de simulation. Certaines d'entre elles sont implémentées si possible dans le modèle de simulation comme l'affectation des moyens de production, le dégroupage/groupage des pièces, etc.

IV.5.2 Définition des Classes d'Objets des Moyens de Production

Les classes d'objets des moyens de production comprennent deux grandes catégories: les **ressources** et les **agents** [YE et al. 94a]. Les ressources tels que outils, outillages et opérateurs, sont tous des objets rentrant dans le fonctionnement d'un atelier hormis les

machines, les transporteurs et les pièces. Elles sont caractérisées par leur exclusivité, partageabilité, leur allocation dynamique, leur aspect consommable, etc. Nous pensons que leurs fonctionnements ou leurs utilisations peuvent être définis implicitement comme des méthodes normales qui seront appelées par des messages des classes "agents". Cette simplification ayant pour objectif de réduire la complexité du modèle et des algorithmes de simulation, en minimisant et en rationalisant le travail de modélisation.

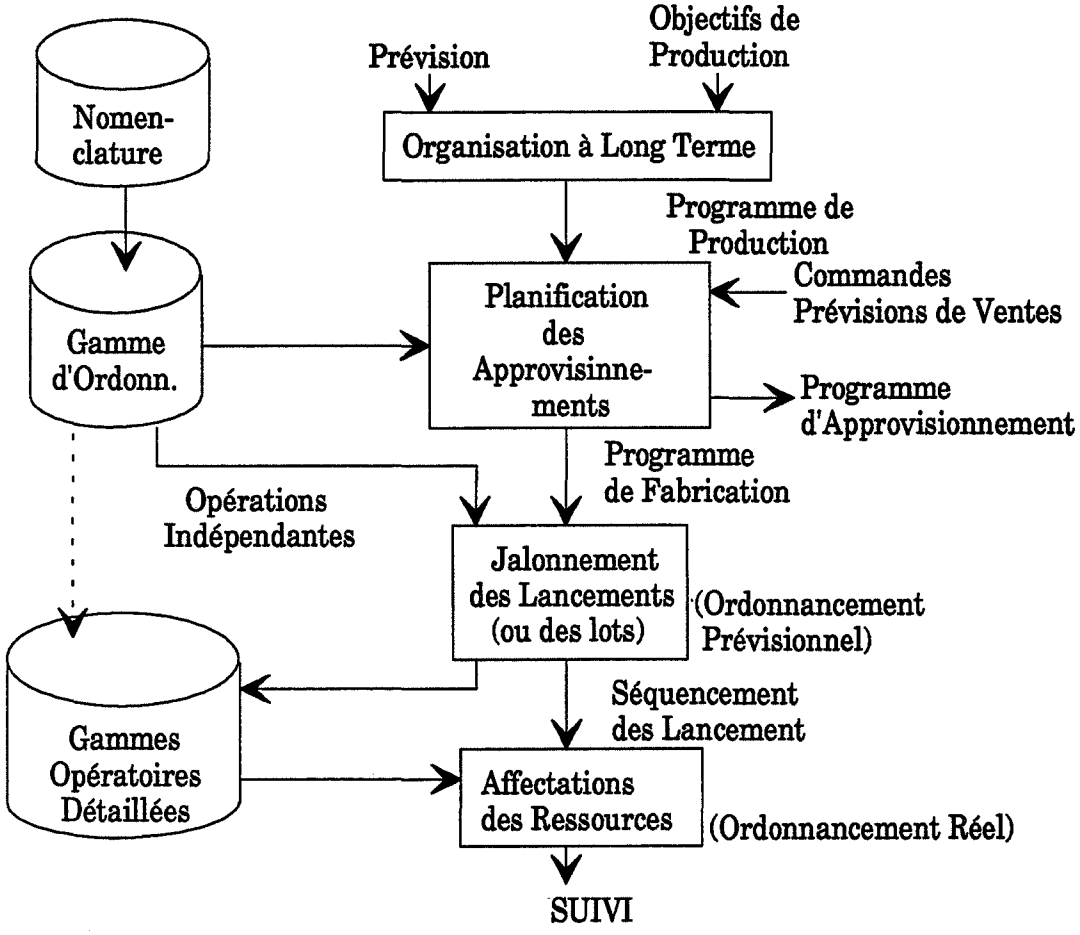


Figure 4-14 Schéma du Système de Commande de la Production

Les objets "agents" comprennent les machines, les stations, les transporteurs et les règles de conduite d'atelier. Ils sont tous dérivés d'un objet "Process" dont l'activité est autonome. Les règles de conduite bloquent et débloquent des processus physiques, ce sont des processus logiques modélisés selon un modèle d'atelier concret. Les machines, les stations et les transporteurs sont modélisés comme des processus physiques (ou comme des processeurs virtuels), ils absorbent des entités en entrée et les restituent en sortie après un délai dépendant de leur comportement et des entités traitées.

IV.5.2.1 Les Ressources

L'étude des différents types de ressources au sens le plus général a permis de dégager 4 critères qui permettent de modéliser leurs propriétés, tant en ce qui concerne leur nature que l'utilisation qui en découle. En voici les caractéristiques:

- **Exclusivité**. Une ressource exclusive ne peut être allouée à un instant donné qu'à une seule des opérations qui en ont besoin. En d'autres termes, elle ne peut être allouée à une opération qu'après avoir été libérée par l'opération précédente.

- **Partageabilité**. Une ressource partageable signifie qu'elle peut être affectée à plusieurs opérations (tâches). Ces affectations peuvent être statiques ou dynamiques (l'affectation peut être modifiée en cours de simulation). Une affectation de ressource partageable est exclusive. En fait, une ressource peut être à la fois exclusive et partageable. L'exclusivité a le trait aux opérations alors que la partageabilité est liée aux agents (équipements) et aux pièces (objets transactionnels).

- **Allocation Dynamique**. Quand l'utilisation de la ressource est terminée, elle peut être allouée pour un autre usage et, si besoin, être transportée vers son nouveau lieu d'affectation.

- **Consommable**. Une ressource consommable (énergie par exemple) est susceptible de s'épuiser et peut nécessiter alors un certain délai pour être de nouveau utilisable. La ressource est alors réinitialisée à une quantité donnée fixe ou aléatoire. Une ressource non consommable (outillage par exemple) peut causer un verrouillage mortel (deadlock) avec une autre ressource non consommable. De plus, une telle ressource a une certaine probabilité de tomber en panne et un certain délai de réparation. Les ressources consommables peuvent être optimisées par leur quantité et leur fréquence de réapprovisionnement, les ressources non consommables par leur nombre et leurs algorithmes de gestion.

Une ressource autonome se déplace elle-même vers le lieu où elle est demandée. Dans ce cas, il vaut mieux la représenter comme un agent. La distinction des moyens de production en ressources et agents est donc subjective, elle dépend de la finesse et de l'objectif de la modélisation. D'autres critères tels que la mobilité de la ressource ou le mode de classement des requêtes peuvent être ajoutés, nous nous intéressons aux trois premiers critères et nous les caractérisons comme des attributs généraux de l'objet ressource.

La gestion des ressources peut être supervisée par un *allocateur de ressource*, qui s'occupe d'organiser les différentes *requêtes* (messages) qui demandent l'allocation. Une requête contient la durée d'utilisation prévue, le nombre d'unités de ressources, le lieu où la ressource doit être

mise à sa disposition (c'est-à-dire l'emplacement du demandeur) et une priorité déterminée par le demandeur et par l'opération à réaliser. L'allocateur de ressources (qui est un algorithme d'une file d'attente des ressources) commence par vérifier si le demandeur n'a pas déjà fait l'objet d'une allocation statique de ce type de ressource. S'il existe un tel lien, l'allocateur répond à celui-ci en désignant la ressource, avec une date de disponibilité immédiate. Sinon, l'allocateur doit chercher d'éventuelles ressources libres du type demandé. S'il en existe, l'allocateur doit en sélectionner une (au sens d'un critère codé dans son algorithme). S'il n'y a plus de ressource libre du type demandé, alors l'allocateur de ressources en avertit le demandeur, en lui indiquant éventuellement une date de disponibilité au plus tôt, calculée d'après la durée d'utilisation prévue des requêtes en cours de service. L'allocateur ne s'occupe pas de la réaction du demandeur (un objet actif) qui est codée dans les méthodes concernées du demandeur. Lorsque l'utilisateur d'une ressource libère celle-ci, il en avertit l'allocateur, qui explore alors sa liste de requêtes afin de tenter d'en satisfaire un nouveau demandeur.

IV.5.2.2 Les Agents

Ferber [FERBER et al. 88] a défini un agent comme une entité physique ou abstraite, qui est capable d'agir sur elle-même et sur son environnement, qui dispose d'une représentation partielle de cet environnement, qui, dans un univers multi-agents, peut communiquer avec d'autres agents, et dont le comportement est la conséquence de ses observations, de sa connaissance et des interactions des autres agents. Les agents sont, dans notre modèle de simulation, implémentés comme des objets actifs dérivés d'un objet "Process" et ont des comportements relativement autonomes. On appelle aussi des objets contrôles ou autonomes. La notion de Process sert principalement à modéliser les moyens de production qui effectuent les opérations sur les pièces. L'objet Process est donc caractérisé par une file d'attente en entrée pour arranger les pièces arrivées (éventuellement une liste d'opérations acceptables par ce processus), par un processus qui décrit le changement d'états des pièces représentant les opérations qu'il est capable d'effectuer et le changement d'états du processus qui modélise les transitions possibles d'états d'un processus, et par une file d'attente en sortie pour déposer les pièces transformées. Un objet Process a besoin d'une autre file d'attente (figure 4-15) pour coopérer (synchroniser et communiquer) avec d'autre processus (c'est-à-dire pour traiter et stocker les messages directs) et une référence des ressources associées (lien statique).

Toutefois, un objet Process peut n'effectuer aucune opération de transformation sur les pièces, pour pouvoir modéliser par exemple une opération de transport, un stock ou une règle de pilotage des opérations. L'objet Process sert donc à modéliser les moyens de transformation (machines, stations, postes de travail, etc.), les moyens de transport (chariots, convoyeurs, transporteurs, etc.), les règles de conduite d'atelier, etc.

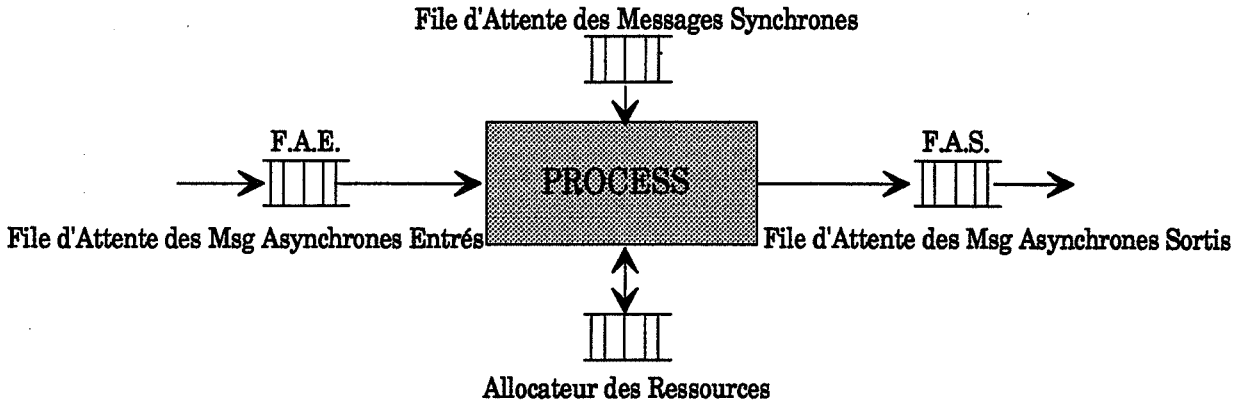


Figure 5-14 Structure du Process (Objet Actif du Point de Vue Processus)

IV.5.2.2.1 Types des Méthodes

Nous avons construit les trois modèles des objets actifs (modèle informationnel, modèle de communication et modèle de transitions d'état) dans le chapitre précédent. Nous détaillons donc ici le quatrième modèle: *modèle de processus*. Avant de détailler, pour chaque objet actif (agent), ses processus ou méthodes qui figurent dans le modèle des transitions d'état, il faut bien étudier le modèle de communication pour définir son comportement: la méthode *fonctionner (ou exécuter)*. C'est une méthode virtuelle sans paramètre qui doit être surchargée par les utilisateurs; puisque chaque agent a un comportement différent des autres. Mais nous pouvons construire des macro-comportements dans la méthode *fonctionner* qui appelle des méthodes virtuelles ou abstraites qui seront définies ultérieurement par les utilisateurs.Par exemple, la méthode *fonctionner* (comportement temporel) d'une machine peut être schématisée comme une boucle infinie (figure 4-16).

Quatre types de méthodes peuvent être distinguées dans la méthode fonctionner d'un agent: les méthodes accès, les méthodes générateurs d'événements, les méthodes transformations et les méthodes tests. Ces types de méthodes sont plutôt caractérisés par leur nature d'utilisation.

Méthodes Accès

Une méthode accès est une procédure dont l'objectif est d'accéder aux valeurs d'un objet. Elle peut être parmi les suivantes:

- Une méthode **Créer** (constructeur), crée une instance d'une classe d'objet,
- Une méthode **Lire,** lit les valeurs des attributs d'un objet,
- Une méthode **Ecrire,** met à jour les valeurs des attributs d'un objet,
- Une méthode **Supprimer** (destructeur), efface une instance d'une classe d'objet.

```
Machine::fonctionner()
Répéter
    Si le stock amont est vide
        bloquer et sauvegarder l'état de la machine et rendre le processeur au scheduler
    Si le stock amont est plein
        réveiller la machine bloquée par ce stock
    Finsi
    Prendre l'article dans le stock amont et acquérir les ressources nécessaires
    Transformer l'article et mettre à jour les informations
    Si le stock aval est plein
        bloquer et sauvegarder l'état de la machine et rendre le processeur au scheduler
    Si le stock aval est vide
        réveiller la machine bloquée par ce stock
    Finsi
    Expédier l'article dans le stock aval et libérer les ressources acquises
A l'infini
```

Figure 4-16 Méthode "Fonctionner" (Macro-Comportement) d'une Machine

Une méthode *accès* est assignée à un message destiné à fournir une (des) valeur(s) d'un objet. Les méthodes *accès* sont en général les plus réutilisables, non seulement dans la méthode *fonctionner*, mais également à l'intérieur de chaque méthode (service) qui figure dans le modèle des transitions d'état. Ces méthodes sont en général publiques et sont: 1) appelées de manière synchrone (non bloquante) par d'autres objets et 2) définies pour tous les objets et indépendantes du changement d'état figuré dans le modèle des transitions d'état d'objets.

Méthodes Générateurs d'Evénements

Un générateur d'événements est une méthode qui produit un événement destiné à un objet récepteur précis ou à un ensemble d'objets récepteurs. C'est une méthode dont 1) la réponse d'événement est très différente selon l'état de l'objet récepteur, 2) cet événement peut perturber le comportement qui figure dans le modèle des transitions d'état et 3) elle peut être appelée de manière synchrone ou asynchrone (communication bloquante), dépendant de la conception du modèle de communication. L'objet qui génère l'événement synchrone attend immédiatement la réponse de l'objet récepteur. Il y a donc un retour du message. Dans le cas d'un événement asynchrone, l'objet qui génère l'événement continue son exécution et cet événement sera pris en compte par l'objet récepteur en cas de besoin.

Méthodes Transformations

Une transformation est une méthode dont l'objectif est un calcul, une transformation des données ou une activité. Dans les deux premiers cas, cette méthode consiste à convertir des données d'entrée en des données de sortie (bien sûr sous une autre forme). Dans le troisième cas, la transformation va durer un certain moment et va provoquer éventuellement des changements d'état d'objets. La méthode transformation peut utiliser ou appeler des méthodes accès pour lire ou écrire les valeurs d'attributs d'objets; elle peut aussi générer des événements synchrones ou asynchrones. Ces méthodes seront raffinées dans le modèle de processus.

Méthodes Tests

Une méthode test permet de faire un choix de direction dans le programme et de passer le contrôle à l'instruction choisie, ou d'établir des boucles en cas de problème d'itération. Cette méthode est normalement utilisée pour définir la précondition de la transformation ou de la génération d'événements. Dans le modèle des transitions d'état, quand l'objet passe d'un état à l'autre, il y a en général une méthode test associée.

Dans un modèle à objets, les objets interagissent par le biais d'événements et de méthodes accès. Ces deux types d'interactions n'ont pas la même nature dans le temps. Quand un objet génère un événement, l'objet récepteur peut ou non prendre en compte cet événement. On peut donc définir des communications asynchrones entre les objets. Par contre, quand un objet accède aux valeurs des attributs d'un autre objet, l'accès aux données se fait en même temps (conceptuellement) que l'exécution de la méthode de l'objet récepteur. On appelle cela une communication synchrone entre les objets. Nous détaillons dans la suite les méthodes bloquantes (processus) par quelques exemples comme les méthodes d'acquisition d'un (des) article(s) et d'expédition d'un (des) article(s) dans la machine et la station.

IV.5.2.2.2 Définition des Processus d'une Machine

Dans la figure 4-15, nous avons construit la structure de base d'un processus. Une machine est un objet dérivé de l'objet "Process". Elle prend l'article (pièce) dans la file d'attente d'entrée (FAE). Si la FAE est non vide, la machine va en choisir un selon une règle de priorité définie par l'utilisateur et va charger les ressources nécessaires selon un algorithme d'allocation de ressources. Si la FAE est vide ou les ressources demandées sont non-disponibles, la machine passe à l'état *arrêt structurel* et l'activité de la méthode *recevoir* est bloqué. La méthode d'acquisition d'un article (*recevoir*) est schématisée dans la figure 4-17. Notons que cette méthode s'exécutera après la méthode *expédier* et dès que cette méthode se termine, elle lance la méthode *transformer*. Cette méthode va générer un événement (un message de déblocage)

vers la (les) machine(s) qui alimente(nt) cette file d'attente si elle est pleine. Différentes règles de décision (objets décisionnels) peuvent être intégrées dans le déroulement des activités de ce processus:

- sélection d'une pièce à fabriquer (choix d'un mode de gestion de file d'attente),
- affectation des ressources appropriées (choix d'un algorithme),
- etc.

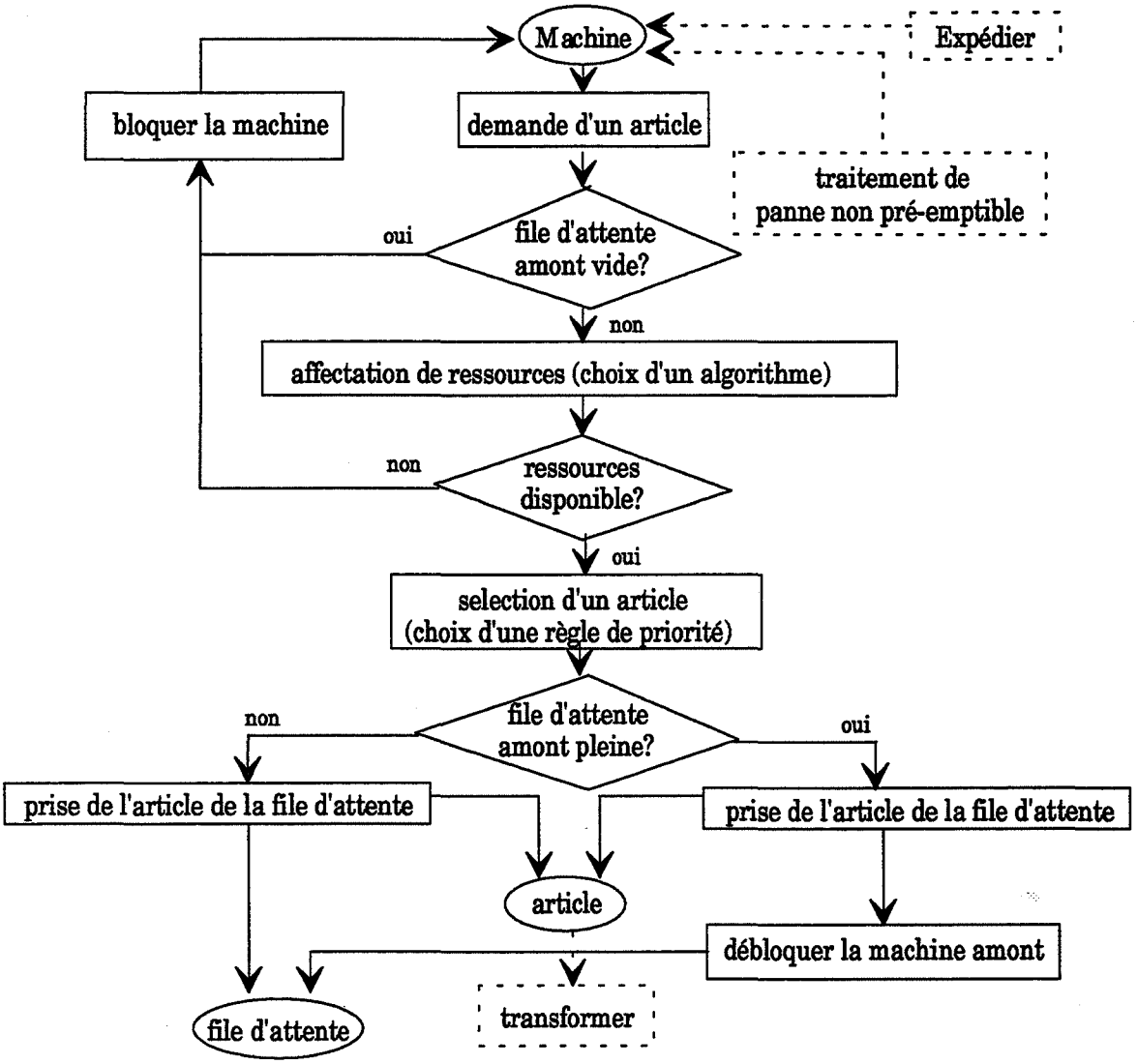


Figure 4-17 Méthode d'Acquisition d'un Article par la Machine :: Recevoir (Item*)

Après le processus de transformation (méthode *transformer*), la machine va expédier l'article transformé dans la file d'attente de sortie (FAS). Si la FAS est pleine, l'activité du processus *expédier* est bloqué, sinon, la méthode met l'article dans la FAS et génère un événement vers la(les) machine(s) qui consomme(nt) des articles dans cette FAS si elle est vide. Diverses

méthodes "accès" utilisées par le processus d'acquisition et d'expédition d'articles peuvent être identifiées dans le modèle de communication de la machine. La figure 4-18 schématise la méthode expédition.

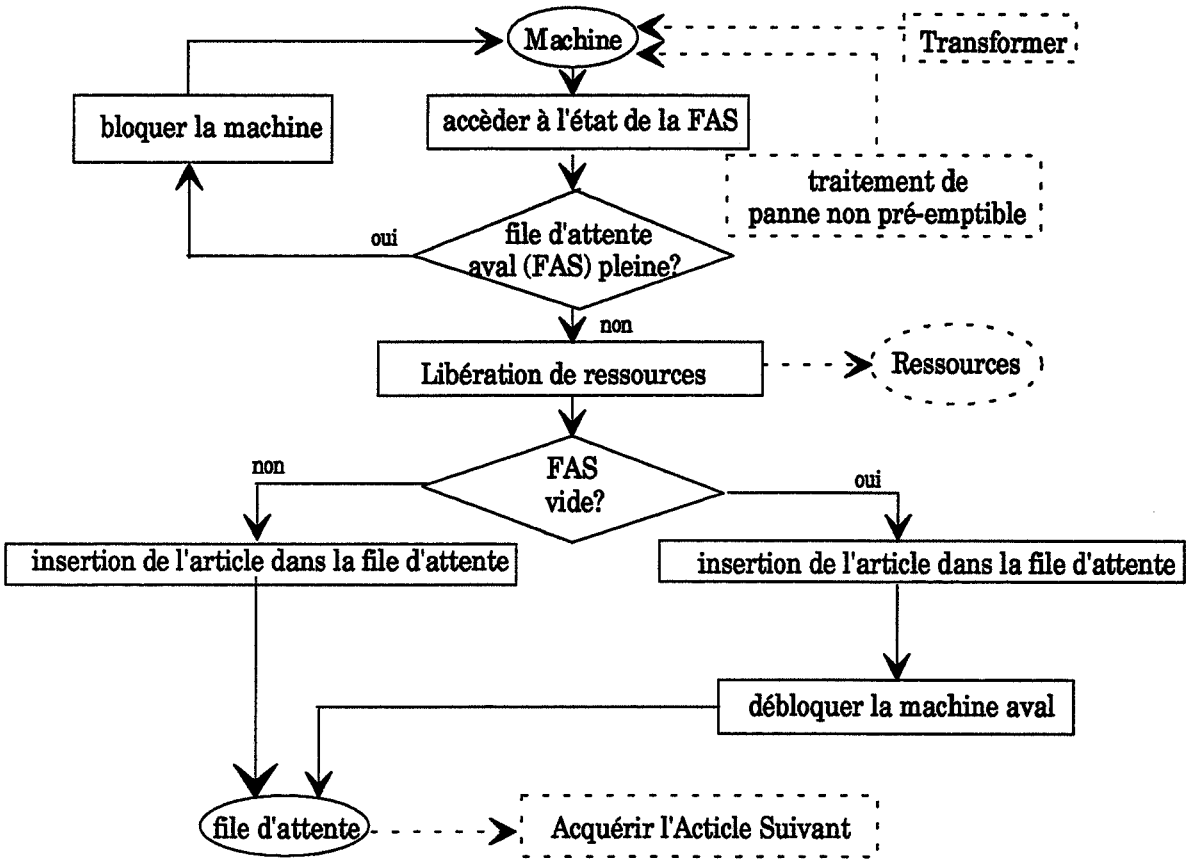


Figure 4-18 Méthode d'Expédition d'un Article par la Machine: expédier (Item*)

Une machine peut être en panne. Deux types de panne peuvent être modélisés [YE et al. 92a]: panne sur une opération *préemptible* et panne sur une opération *non préemptible*. Une panne sur l'opération non préemptible (figure 4-19) spécifie que le processus panne n'arrive pas pendant le processus transformation, et que la mise en panne surviendra immédiatement avant ou après la transformation. Une panne sur l'opération préemptible (défaillance) spécifie que la machine doit être immédiatement retirée de l'activité production (processus transformation), et la méthode associée doit décider soit de terminer le processus transformation (s'il arrive pendant la transformation), soit de la suspendre. Si la transformation est suspendue, une méthode décide soit d'attendre la disponibilité de la même machine (comme un arrêt structurel) soit d'accepter un remplacement (en véhiculant les articles en attente devant une autre machine). Ces méthodes décisionnelles sont conçues comme des méthodes virtuelles et seront surchargées et précisées lorsque l'utilisateur instancie l'objet machine. La définition du processus transformation est très différente selon le type de panne. Pour une opération non préemptible, les traitements de pannes définis dans le processus *transformer* sont implémentés

par deux appels à une méthode virtuelle de gestion de pannes. Par contre, pour une opération préemptible, l'utilisateur doit surcharger le processus "transformer" afin d'intégrer sa propre gestion de panne.

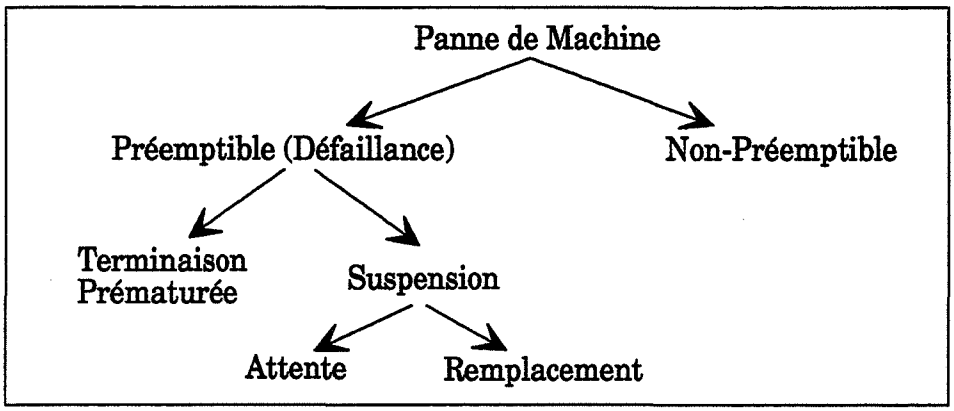


Figure 4-19 Gestion de Panne sur le Processus Transformer d'une Machine

IV.5.2.2.3 Définition des Processus d'une Station

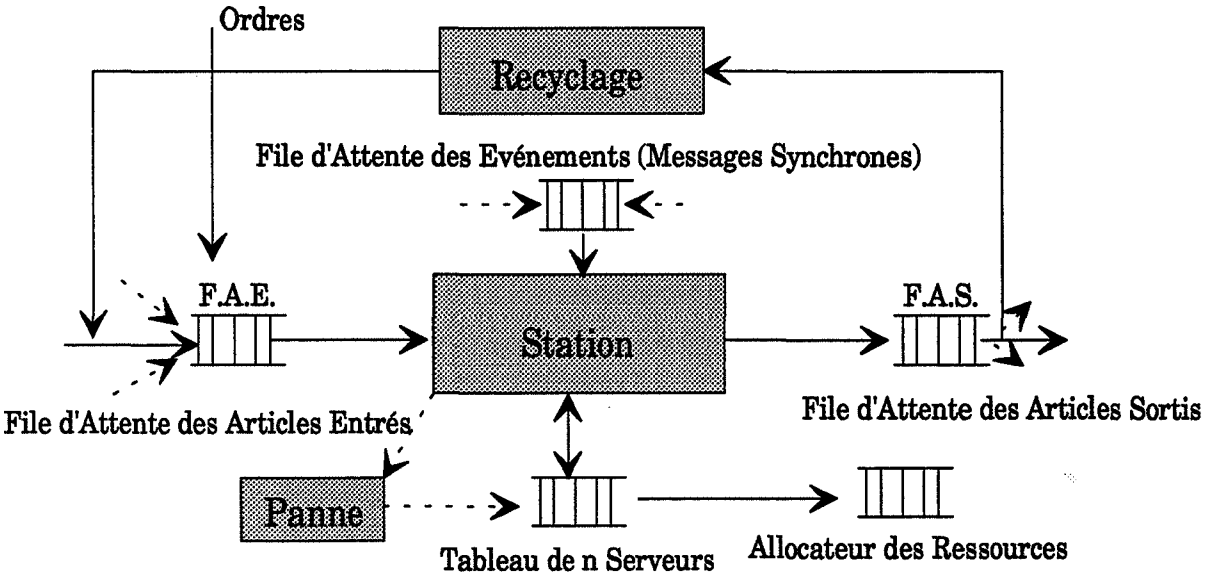


Figure 4-20 Structure Générale d'une Station

Comme nous l'avons constaté, il y a deux possibilités pour implémenter les processus d'une station qui contient n serveurs: un processus par serveur ou un processus pour tous les serveurs. Nous adoptons la deuxième approche pour les problèmes de temps de calcul (performance) et une meilleure utilisation des concepts d'objets: l'encapsulation et l'abstraction. Pour simplifier la présentation, nous n'abordons pas ici la gestion des ressources et la gestion de panne des serveurs. La structure générale d'une station (figure 4-20) est très similaire à celle

d'une machine, mais se distingue par un pointeur sur un tableau de serveurs. Les processus de la station servent à simuler les processus de ces n serveurs qui travaillent en parallèle.

Si la file d'attente amont contient des articles et si la station a des serveurs libres, elle va charger ces derniers (méthode *recevoir*) jusqu'à ce qu'il n'y ait plus d'articles dans la file d'attente amont ou jusqu'à ce qu'il n'y ait plus de serveurs libres. Si la file d'attente est vide et tous les serveurs sont libres, la station est bloquée et elle va être débloquée par un message "déblocage" de la station amont qui alimente la file d'attente. Sinon, elle déclenche le processus *transformer*, c'est-à-dire, les serveurs chargés sont à l'état productif et l'horloge est avancée par une durée déterminée. La figure 4-21 schématise le processus d'acquisition (méthode *recevoir*) d'articles.

Le processus *transformer* de la station simule les serveurs chargés qui travaillent en parallèle. A un moment donné, une station peut se trouver dans un des cas suivant: tous les serveurs sont chargés, certains serveurs sont libres, certains serveurs attendent d'être déchargés ou certains serveurs sont en panne, tous les serveurs sont libres, etc. Les serveurs n'ont pas donc les mêmes rythmes de travail ou leurs performances sont différentes. Pour encapsuler la gestion du temps de transformation de ces serveurs, le processus transformation de la station va continuer jusqu'à ce qu'il n'y ait plus de serveurs chargés. L'algorithme de transformation d'une station peut se définir de la manière suivante:

```
Station::Transformer
Répéter
    pour les serveurs chargés (un ensemble t)
        calculer la durée de transformation commune 'd' de ces serveurs
        trouver les serveurs (un ensemble n) dont la durée de transformation est égale à 'd'
        avancer le temps d'horloge du système de 'd' (delay (d))
        changer l'état de l'ensemble de serveurs n (de l'état productif à l'état prêt-à-décharger)
        mettre à jour la durée restante de transformation pour l'ensemble de serveurs (t-n)
    fin pour
A l'infini
```

Le processus de gestion de panne d'un serveur est défini différemment de celui d'une machine. Un serveur n'est plus un agent dans la station. Il est défini comme un objet passif dont le comportement est encapsulé dans les méthodes de la station. La gestion d'une panne est donc similaire à la gestion d'une ressource. On peut retirer un serveur du tableau de serveurs de la station pendant le temps de panne. L'utilisation des ressources est associée au serveur, il n'y a plus de lien direct entre l'agent station et les ressources qui sont manipulées par les méthodes

de la station. La station accède à ces ressources à travers ses serveurs. Avec cette structure, nous pouvons ajouter des règles de pilotage des serveurs ou des choix d'itinéraire des objets circulants entre les serveurs.

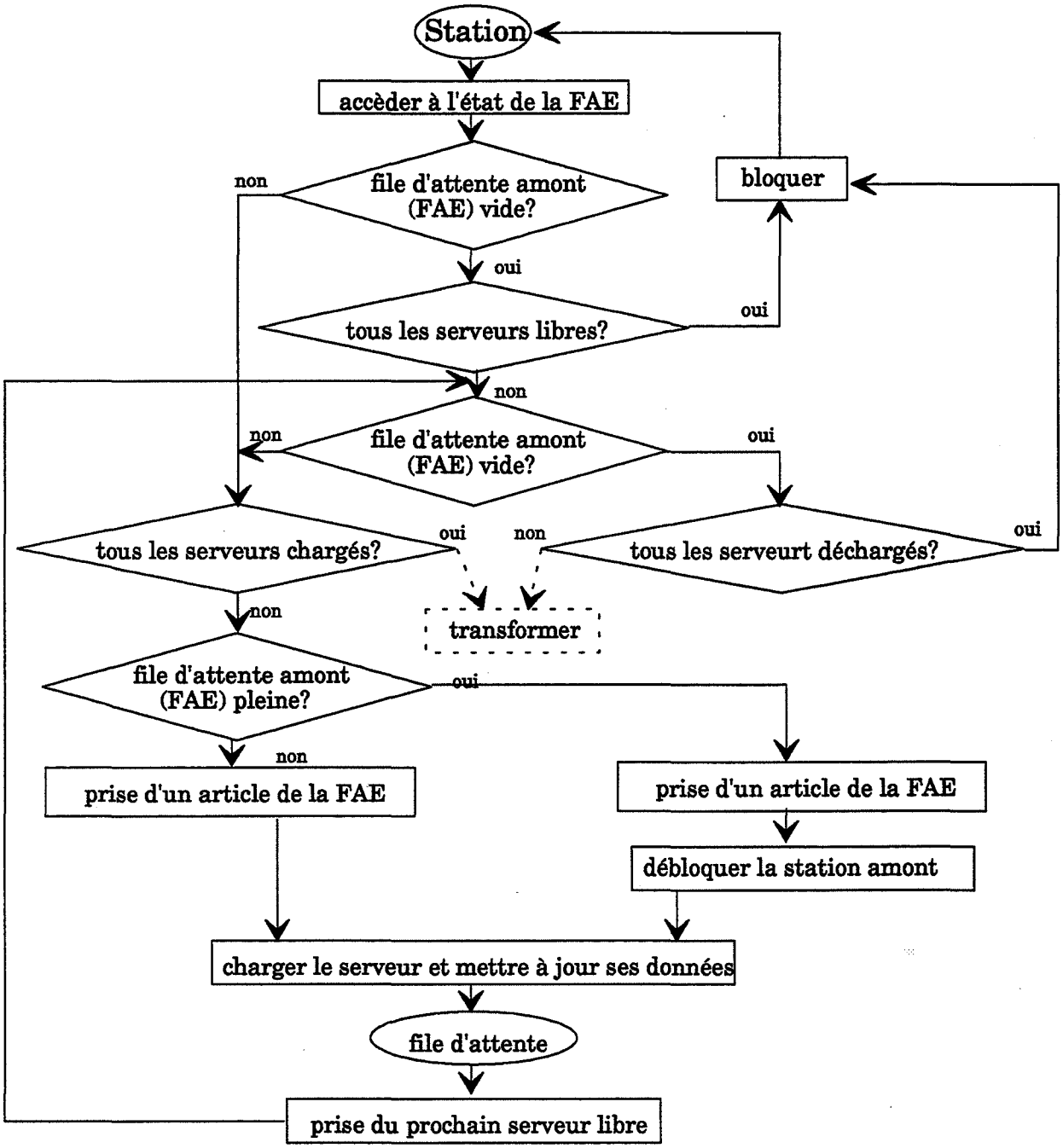


Figure 4-21 Diagramme de la Méthode Station::Recevoir (Item**)

Le processus d'expédition (méthode *expédier*) des articles transformés d'une station est schématisé dans la figure 4-22. Le processus va continuer jusqu'à ce qu'il n'y ait plus de serveurs en état *prêt-à-décharger* ou jusqu'à ce que la file d'attente aval soit pleine dans le cas

où la capacité de file d'attente est limitée. Dans le cas où la file d'attente est vide, la station va générer un message de déblocage pour la(les) station(s) aval qui consomme(nt) des articles de cette file d'attente.

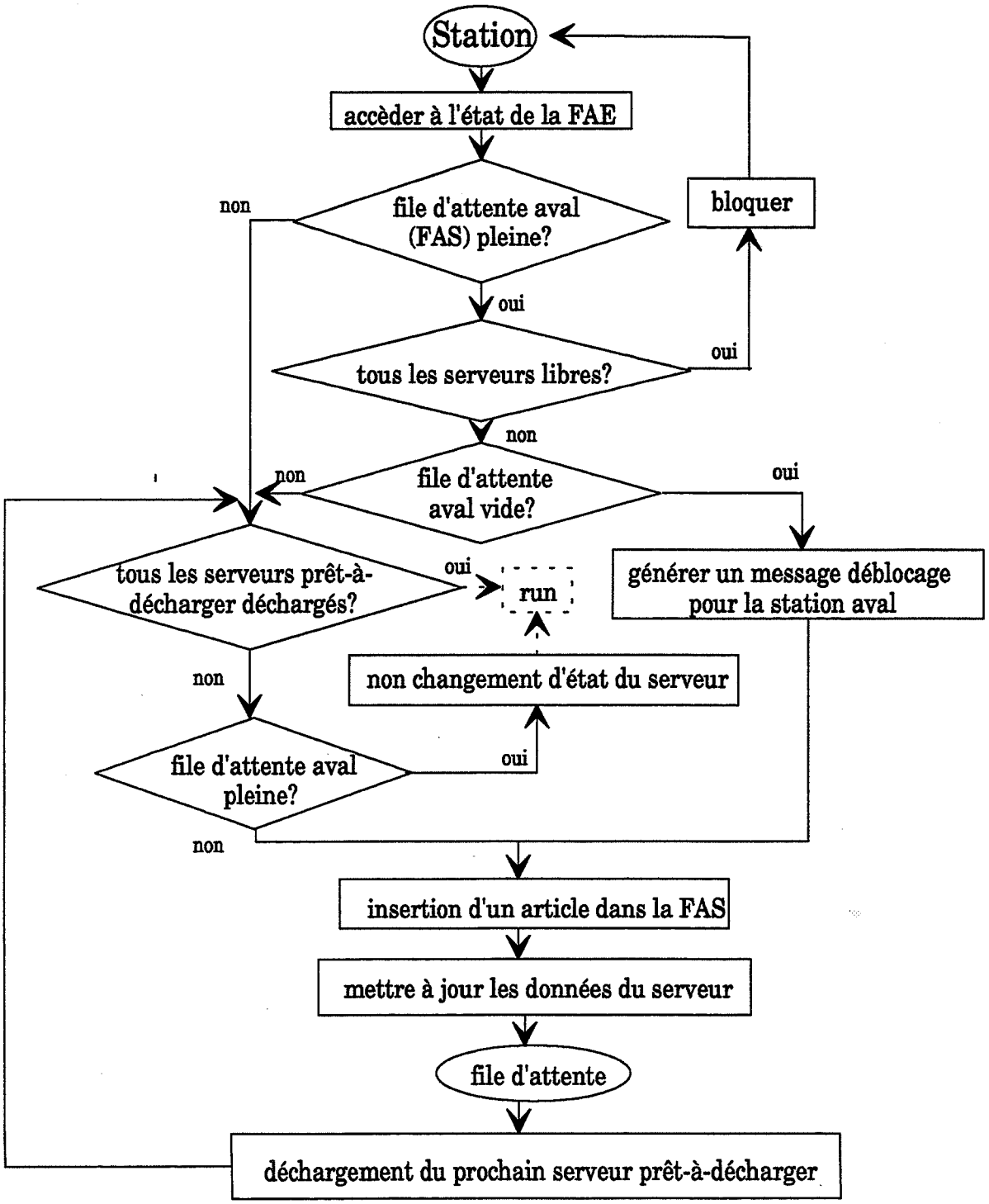


Figure 4-22 Diagramme de la Méthode Station::Expédier (Item**)

De la même manière, nous pouvons définir, à l'aide de ces processus élémentaires, les processus des objets actifs (agents) des systèmes de production comme un convoyeur ou un transporteur, une machine avec des stocks à capacité infinie, une ressource autonome, un processus de pilotage global, etc. La figure 3-10 du chapitre III, par exemple, illustre un comportement d'un objet commande si l'on le modélise comme un objet actif.

IV.5.3 Définition des Classes d'Objets Décisionnels

Les classes d'objets décisionnels des systèmes de production constituent l'instrument le plus général, le plus ouvert et le plus puissant pour piloter l'atelier. Elles permettent d'exprimer n'importe quel programme de pilotage, au sens informatique du terme, qu'il soit issu d'une réflexion effectuée au moment de la conception du système de production ou qu'il provienne de l'expérience acquise par le responsable de l'atelier. Cet aspect d'enrichissement n'est pas le seul avantage car il est très probable que la conception des systèmes de production ne peut fournir une solution figée aux impératifs de production. Ainsi il est nécessaire d'avoir ces objets décisionnels exprimés clairement et modifiables comme des entités à part entière.

L'adaptation de l'atelier au contexte de production (réactivité) est un des objectifs clé de cette recherche (flexibilité du modèle). Elle doit se traduire par une possibilité d'action (choix par exemple) des règles ou des algorithmes de pilotage. L'élaboration et le choix de ces règles et de ces heuristiques est un point délicat, mais ce n'est pas la seule difficulté. En effet, la modélisation et la simulation doivent être utilisées d'une manière efficace, nous avons renoncé à implanter ces règles comme base d'un moteur d'inférence interrogeable par les algorithmes de simulation ([JAIN et al. 90], [YE et al. 92a], [GUO et al. 90]); et nous avons dû traduire ces règles en des algorithmes et des procédures, susceptibles d'être invoquées directement par les méthodes des objets agents ou les algorithmes de simulation.

Les classes d'objets sont composées de règles de gestion et de règles de pilotage du système. Trois types de règles doivent être modélisées: les règles de priorité (dispatching), de management et de gestion des ressources [GUILLARD 92].

IV.5.3.1 Les règles de Priorité (Dispatching)

Les règles de priorité consistent à organiser le choix des opérations (articles) sur lesquelles vont travailler les agents (machine, station, etc.), et réciproquement, pour chaque opération choisie, l'agent qui va la traiter dans la mesure où il y en a plusieurs possibles (station). Ceci conditionne aussi la circulation des agents transporteurs. Le travail de répartition se fait dans la mesure des libertés laissées par la structure des gammes opératoires associées aux articles et par le potentiel opératoire des agents dans l'atelier.

Ces règles ont un caractère *local*, puisqu'elles concernent des choix ne portant que sur l'état des opérations, des agents de transfert qui devront les prendre en charge et des agents machines ou stations ayant le potentiel opératoire nécessaire. L'état du reste du système influe peu ou nullement sur le fonctionnement de ces règles. De plus, ces règles ont un caractère *anonyme*, puisqu'elles peuvent s'exprimer sans désigner explicitement les objets concernés. Par exemple, on peut avoir une règle du genre: "A la sortie d'une opération d'assemblage sur une machine, l'article doit aller à la machine de test dont les places d'attente d'entrée sont les moins remplies", ou encore "L'article doit aller sur la cellule flexible qui demandera l'allocation du moins de ressources possibles pour les traiter".

Une centaine de ce genre de règles est fournie dans la littérature [PANWALKER 77]. Chaque atelier peut appliquer une règle ou des règles combinées mieux adaptées à son cas particulier. Ici réside la difficulté majeure du problème: une règle peut donner d'excellents résultats dans certains cas et de très mauvais dans d'autres. Aussi, en dynamique, lorsque de nouveaux ordres de fabrication arrivent, et donc changent des données (contexte), les performances obtenues en appliquant toujours la même règle peuvent se dégrader. C'est pour cela que nous abstrayons des heuristiques générales en des objets modifiables et fournissons aux utilisateurs un modèle ouvert et dynamique de simulation qui leur permet d'appliquer ces heuristiques, directement ou par enrichissement de leurs connaissances. Le tableau 4-1 présente les règles de priorité les plus utilisées [CANALS 86]:

Règles	**Commentaire**
FIFO	premier arrivé premier servi
SPT	plus petit temps opératoire
EDD	la date au plus tard (d_i) l'article qui contient l'opération
SLACK	plus petite marge entre la d_i et temps courant
SLACK/OPN	plus petite marge par opération
SPT-T	mixage de SPT et de SLACK/OPN
SPT-X	mixage de SPT et de SLACK
CoverT	pondération de SPT par le coût du retard
TZAR	marge plus temps opératoire
S	pondération statique de SPT par le temps d'attente
SMOT	opération de l'article de durée moyenne opératoire min.
MOPNR	la plus longue opération de l'article de gamme restante
FOPNR	la plus courte opération de l'article de gamme restante

Tableau 4-1 Règles de Priorité

D'autres règles telles les règles de lancement des ordres de fabrication existent [BOUKACHOUR et al. 93]. On y a souvent recours dans l'industrie. La différence de ces règles est dans la définition de deux méthodes *insérer* ou *enlever* de l'objet décisionnel qui dérive d'une file d'attente. Elles peuvent être très bien définies à partir d'une classe d'objet *file de priorité*: l'utilisateur surcharge donc l'algorithme d'arrangement des articles dans la file et les deux méthodes insérer et enlever de la file pour construire leurs propres règles de priorité. Ces règles sont très attachées à la structure des articles contenus dans la file. Une règle de priorité comme FIFO ou LIFO peut être implémentée sans connaître la structure de l'article. Par contre, une règle SPT doit connaître le temps opératoire de l'article sur cette machine et une règle EDD le temps opératoire restant et la date de sortie souhaitée, etc. Ces règles sont définies comme des méthodes normales d'un objet décisionnel, mais leurs applications sont définies comme des méthodes virtuelles ou méthodes pures dans les objets agents. C'est au moment de l'exécution que les utilisateurs précisent quelles décisions doivent appliquer par les objets actifs dans une application.

IV.5.3.2 Les Règles de Management

Les règles de management (ou règles de comportement global) ont pour rôle de traiter des événements qui peuvent survenir dans l'atelier, au sens large. On exprime ainsi une sorte de programme de contrôle qui traite l'atelier dans son ensemble (bien qu'il puisse fonctionner en particularisant telle ou telle partie). Au contraire des précédentes, ces règles ont donc un caractère *global* et *explicite* (non anonyme). Ces règles proviennent de l'expérience du responsable de l'atelier ou de la simulation. Bien entendu, ceci n'est pas une loi générale. Un exemple pourrait être: "Si la machine n°5 tombe en panne, alors la machine n°3 doit cesser de fabriquer le type de pièce 4 puisqu'il ne sera pas traité; la machine n°3 peut changer le mode de sélection dans sa file d'attente d'entrée ou remettre en cause son comportement".

Ces règles ne concernent que les événements les plus probables et ne peuvent pas traiter exhaustivement l'ensemble des problèmes susceptibles de survenir dans l'atelier (si c'était le cas, on disposerait d'un outil analytique capable de résoudre tous les problèmes de l'atelier). Dans l'exemple proposé, cela revient à dire que la machine n°5 tombe fréquemment en panne.

Nous considérons que ces règles sont basées sur un modèle (général ou spécifique). Quand nous modélisons les classes d'objets logiques et physiques, nous devons fournir des attributs et des méthodes pour répondre à ces messages envoyés par les règles de "management".

IV.5.3.3 Les Règles de Gestion de l'Allocation des Ressources

Les règles de gestion de l'allocation des ressources servent à classer les demandes d'allocation selon un critère particulier. Par exemple, on pourra décider de gérer l'allocation d'un outil (outillage, opérateur, etc.) en se basant sur les priorités des articles à fabriquer ou bien simplement de gérer les requêtes selon leur ordre d'arrivée.

Ces règles proviennent également de la phase de conception du système de production, mais elles conditionnent beaucoup la fluidité de la circulation des objets de transactions et doivent éventuellement être remises en cause et évaluées. Les ressources sont ensuite gérées par un allocateur de classe (voir section V.2.1).

V Construction des Classes d'Objets de Résolution

Nous avons examiné et raffiné les classes d'objets sémantiques (ou classes d'objet du domaine). Un système logiciel à objets contient aussi des classes d'objets de résolution, c'est-à-dire, des classes d'application, des classes d'interface, des classes auxiliaires/mathématiques.

V.1 Classes d'Objets d'Application

Les classes d'objets d'application peuvent être considérées comme les mécanismes de conduite ou de contrôle du système. Les classes d'objets d'application pour la simulation concernent essentiellement la définition des classes d'objets *échéancier*, le noyau de synchronisation et les mécanismes de synchronisation et de communication entre les processus (chapitre V).

V.2 Classes d'Objets Auxiliaires/Utilitaires

Les classes d'objets auxiliaires/utilitaires sont des objets qui sont indépendants des applications. Les objets de base ou auxiliaires comme les listes (liste simple, liste double, arbre, tableau, gestion des fichiers, etc.) sont définies dans la littérature tant au niveau de la spécification qu'au niveau de l'implémentation. Les objets auxiliaires mathématiques comme les matrices, les vecteurs, les différents algorithmes de tri (HeapSort, QuickSort, ShlSort) et les objets utilitaires pour la simulation comme les différents générateurs (poissonnien, exponentiel, gaussien, etc.), les objets d'analyse (variance, moment, espérance, histogramme, lissage exponentiel, etc.) sont aussi prédéfinis par la plupart de progiciels ou d'outils pour la modélisation et la simulation. Nous pouvons les étendre facilement comme des objets d'analyse des performances des systèmes de production, des objets de gestion des informations, etc.

V.3 Classes d'Objets d'Interface

Les classes d'objets d'interface, c'est-à-dire la manifestation visible des objets sémantiques sur l'écran, ne sont pas directement liés aux problèmes à résoudre, mais elles représentent les points de vue des objets sémantiques par l'utilisateur (objets fenêtres, objets boîtes de dialogue, objets contrôles, etc.). La tâche principale dans cette phase de conception est d'élaborer une implémentation graphique des objets sémantiques pour afficher les états de sortie, les masques de saisie, les communications, etc., en utilisant les classes d'objets informatiques de base. Un exemple de configuration d'une machine avant le lancement de simulation, par exemple, est illustré dans la figure 4-23. Cette interface est pour la saisie des informations sur les attributs structurels et conjoncturels de la machine.

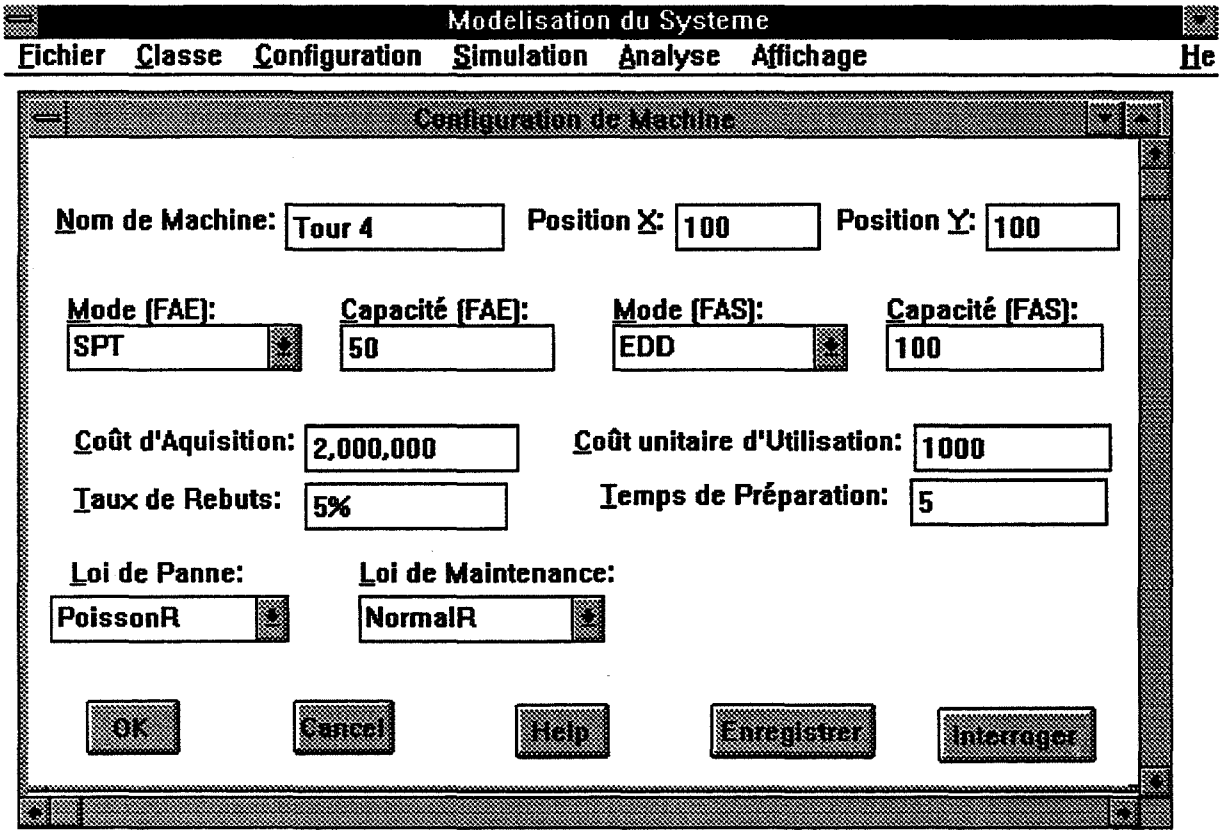


Figure 4-23 Ecran de Configuration d'une Machine

D'autres objets sémantiques d'interfaces tels que le marché, la commande, le programme de production, le produit, la ressource, etc., sont aussi construits avec le package de programmation graphique: ObjectWindows. Une interface (base de données) entre ces informations de configuration est ainsi réalisée.

VI Conclusion

Bien que le nombre de publications concernant les méthodes de conception orientée-objets se multiplie, les thèmes fondamentaux et les approches (en vocabulaire et en notation) de ces méthodes ont plus de similitudes que de différences. Il est difficile (voire stupide) de qualifier ou de quantifier la puissance et la faiblesse de chaque méthode. Notre but est d'essayer de tirer les meilleurs concepts de chaque méthode et de les intégrer dans notre travail. Dans ce travail, nous nous sommes inspiré de beaucoup de techniques telles que nous trouvons dans l'analyse du domaine, la programmation concurrente, le langage de programmation C++, les concepts temps réel et les concepts objets concurrents. La démarche de conception proposée dans ce mémoire a pour objet de construire des classes d'objets implémentable directement en langage C++: une plate-forme de développement pour l'application.

Nous avons détaillé, dans ce chapitre, l'étape de la conception de modèles de simulation des systèmes de production en deux phases: une phase de conception de haut niveau qui consiste à modéliser les classes d'objets du domaine et de son environnement ainsi qu'une phase de conception de bas niveau qui consiste, d'une part à raffiner, à étendre, à hiérarchiser et à réorganiser les classes d'objets du domaine et d'autre part, à définir la structure interne du logiciel tels que l'architecture de modèle, les classes d'objets d'application, les classes d'objets d'interface et les classes d'objets auxiliaires/utilitaires.

Nous avons commencé par la conceptualisation des objets actifs, un concept très important dans l'approche par processus dans les modèles de simulation à objets. Un objet autonome est un objet actif qui possède son propre flot de contrôle (scripts de processus). Il encapsule non seulement sa structure (organisation) interne, mais également fournit des primitives de création de processus et des mécanismes de coopération entre eux. Les problèmes principaux concernent la granularité des activités concurrentes à l'intérieur d'objets: 1) une seule ou plusieurs activités existent dans un même objet; 2) les messages entre objets sont synchrones ou asynchrones (et aussi unidirectionnels ou avec retour); 3) le nombre d'activités est-il fixé ou non (la dynamicité de la configuration des processus), etc. Nous les distinguons en quatre activités dans les objets actifs: perception et acquisition, cognition, décision et action.

Ensuite, nous avons construit une architecture ouverte et flexible qui permet à différents utilisateurs d'y intervenir à différents niveaux des modèles (ressource, atelier, unité de production, GPAO). Les aspects sur les perspectives de la représentation par objets, les points de vue des objets, les relations entre les classes d'objets et le cycle de vie des classes d'objets sont analysés avant de présenter les classes d'objets sémantiques dans la simulation des flux de production: classes d'objets de transactions, classes d'objets des moyens de production et classes d'objets décisionnels. Les attributs et méthodes de ces classes d'objets sont présentés et

conceptualisés. Nous pouvons implémenter directement ces classes d'objets dans un outil de simulation qui supporte la notion des processus "légers" (threads).

Enfin, le quatrième modèle d'objets: les modèles de processus sont illustrés à travers les exemples d'objets de machine et de station, et la conception des objets auxiliaires et des objets d'interfaces est brièvement présentée.

Chapitre V Simulation Concurrente par l'Approche Objet

I Introduction

Les techniques de la programmation concurrente ont considérablement changé durant cette dernière décennie ([ANDREWS 83], [WILLIAMS 90]). Premièrement, les progrès théoriques ont engendré de nouvelles définitions pour la programmation concurrente qui permet d'exprimer des processus concurrents plus simples, de construire des communications et des synchronisations plus explicites, de faciliter la preuve formelle d'exactitude du programme concurrent, etc. Deuxièmement, l'utilisation des systèmes parallèles ou des systèmes répartis permet d'implémenter réellement des programmes concurrents ou parallèles. La programmation concurrente n'est donc plus uniquement la compétence de ceux qui conçoivent ou implémentent des systèmes d'exploitation. Elle devient une notion essentielle et indispensable pour tout type de programmation, tels que les systèmes de gestion des bases de données, les applications en temps réel, les systèmes de contrôle embarqués, les algorithmes parallèles et les simulations à événements discrets bien entendu.

La notion de concurrence ou parallélisme est présente dans toutes ces catégories d'application. Dans chacune d'elles le temps intervient avec une notion différente et d'une façon plus ou moins explicite. En ce qui concerne la simulation, le temps qui intervient est soit un *temps virtuel* [JEFFERSON 83], soit un *temps physique* ("réel") pour des applications réelles [HOOGEBOOM et al. 93]. Le temps virtuel peut être associé au temps physique par des relations plus ou moins complexes. Par exemple, dans le cas de l'assignation des priorités aux processus sur un calendrier, le temps intervient d'une façon implicite. Par contre, dans le cas où un certain processus attend une ressource ou une condition pour déclencher un événement, le temps est spécifié explicitement. Il intervient souvent de manière explicite dans le cas de la simulation à événements discrets.

Les modèles de simulation avec des concepts concurrents ou parallèles peuvent être implémentés de différentes manières [WILLIAMS 90]: modèle séquentiel avec un ou plusieurs processeurs, modèle avec des machines vectorielles, modèle avec des processeurs pipelines, modèle avec une mémoire distribuée, etc. On s'intéresse uniquement, dans notre cas de recherche, à la programmation concurrente avec la notion de processus "léger" (light-weight process) implémenté par un objet abstrait "*thread*" pour la simulation. Les threads ne sont pas un langage mais des bibliothèques de constructeurs. Ils permettent d'avoir plusieurs flots de

contrôle dans le même espace d'adressage. Ils ont leurs propres activités et ressources (piles, données) [WALAS et al. 93].

L'approche orientée-objets (AOO), un nouveau concept pour l'analyse, la conception et la programmation, nous laisse entrevoir beaucoup d'avantages dans certains domaines d'application par rapport aux approches fonctionnelles et structurées. En pratique, n'importe quel objet du monde réel peut être modélisé par un objet informatique dans un système à objets. Comme les objets réels peuvent exister et s'exécuter concurrentiellement, l'approche à objets montre aussi son utilisabilité potentielle pour la construction des systèmes concurrents. Différents types de concurrences existent dans un système à objets: différents objets peuvent exécuter diverses activités (méthodes) en même temps ou un objet peut exécuter plusieurs activités (méthodes) concurrentiellement. Une explosion potentielle des activités concurrentes est donc possible dans un modèle à objets. Comme dans n'importe quel langage concurrent, le problème majeur du langage orienté-objets (LOO) est sa capacité d'exprimer et de contrôler les activités concurrentes: peut-on concevoir des abstractions, dans le sens orienté-objets, réutilisables et extensibles, et qui permettent des utilisations concurrentes comme dans le cadre de la programmation parallèle, et si oui par quels mécanismes syntaxiques les décrire.

La plupart des langages à objets traditionnels ne peuvent pas être considérés comme des langages concurrents bien qu'ils fournissent une certaine forme de concurrence (langages basés sur C avec le concept *coroutine*, langages basés sur Lisp avec le mécanisme de "*diviser et conquérir*", langages basés sur Smalltalk comme DistributedConcurrentSmalltalk et ConcurrentSmalltalk, langages basés sur la notion d'acteur, etc.). Diverses recherches sont en cours dans ce domaine ([NELSON 91], [WYATT et al. 92]). Nous présentons d'abord les concepts du processus en terme d'implémentation et essayons d'éclaircir les concepts "concurrence" (création de processus, synchronisation et communication entre processus). Ensuite, nous analysons les trois langages (ou logiciels) de simulation à objets qui supportent ces concepts et qui sont disponibles dans notre laboratoire. Enfin, en utilisant un logiciel de simulation Meijin++, nous détaillons comment nous implémentons des modèles de simulation du système de production par des entités communicantes dont chacune est un objet actif.

II Définitions

II.1 Processus

Au concept de traitement parallèle, il faut attacher un certain nombre de notions fondamentales dont celles de **processus** et de **procédure**. Une procédure (ou un programme) est une entité

composée d'une ou de plusieurs séquences d'instructions, agissant sur un ensemble de données (les variables externes ou internes à la procédure). Une procédure est essentiellement *statique*.

Un processus, entité essentiellement dynamique, met en oeuvre de manière séquentielle une ou plusieurs procédures statiques, en vue de la réalisation d'une activité donnée par un processeur. Notons qu'une *activité* est une action de processus (qui se déroule donc dans un laps de temps ni négligeable ni indifférent a priori).

Avant de continuer, précisons les notions de *tâche* et de processus. Certains puristes font une distinction entre tâche et processus en insistant particulièrement sur l'aspect *cyclique* de l'exécution d'une tâche tandis qu'à un processus sont plutôt attachées des notions de *progression* et d'*action*. Cette distinction n'a que peu de conséquences pour la compréhension de la suite.

II.1.1 Contexte de Processus

Un processus ou encore une tâche, est fonction du temps; il peut être *créé*, *exécuté*, *bloqué* ou encore *détruit*. Un processus exécute les instructions du programme auquel il est associé. Sans entrer dans les détails, un processus comporte différentes zones, en général trois, qui constitue une partie du *contexte* d'un processus:

* Une zone *programme* contenant les instructions du programme et éventuellement certaines constantes (coefficients, texte, etc.), considérée par le système comme une zone constante accessible uniquement en lecture, si l'entité de gestion de la mémoire le permet.
* Une zone de *données* contenant les données variables et les données constantes ou initialisées. Cette zone est bien sûr accessible en lecture et en écriture.
* Une zone de *pile* permettant de ranger des informations temporaires (variables locales, adresses de retour de sous-programme, etc.) accessibles également en lecture et en écriture.

L'accès aux différentes zones s'effectue généralement par rapport à des registres initialisés par le système lui-même lors du lancement du processus. Ainsi, le compteur de programme est initialisé sur la première instruction du programme, la zone de données et la pile étant, quant à elles, respectivement référencées par un pointeur de zone de données (registre de donnée ou d'index) et un pointeur de pile. Ces informations font partie de ce que l'on appelle le **contexte** du processus. En d'autres termes, ce contexte comporte généralement:

* l'état des registres du processeur;
* l'état des pointeurs de zones mémoire associées au processus (instructions, données, pile);

* les caractéristiques du processus dans lesquelles on retrouve essentiellement: un *identificateur* grâce auquel l'ordonnanceur ou le système d'exploitation identifie ce processus, une *priorité* permettant de quantifier le degré d'urgence du processus (Cette priorité permet de déterminer l'ordre d'allocation du processeur lorsque plusieurs processus sont en attente d'exécution.), un *droit d'accès* spécifiant les ressources matérielles ou logicielles accessibles par le processus, un *état* instantané du processus, etc.

II.1.2 Etats du Processus

Un processus créé dans un modèle simplifié de simulation ne peut être que dans l'un des quatre états fondamentaux suivants:

- **non-opérationnel (créé, dormant, hors-service)**: le processus est créé et connu de l'ordonnanceur (voir section II.1.5) (c'est-à-dire le contexte du processus est initialisé et maintenu par l'ordonnanceur),
- **élu (en cours, en exécution)**, le processus est en possession d'un processeur et en cours d'exécution,
- **éligible (prêt)**, le processus est en possession de toutes les ressources nécessaires à son fonctionnement sauf d'un processeur,
- **bloqué (en attente)**, le processus est en attente d'une ressource quelconque indispensable à son exécution future.

Il est important de distinguer l'état **éligible** de l'état **bloqué**. Lorsqu'un processus passe en exécution à partir de l'état éligible, il suffit de mettre à jour une variable signalant le nouvel état du processus et d'initialiser les registres du processeur. Pour reprendre un processus bloqué, il faut recharger tout le contexte sauvegardé lors du blocage. Le processus reprend alors exactement à l'endroit interrompu. Les transitions entre états du processus sont précisées sur la figure ci-après (figure 5-1):

- la transition non-opérationnel -> éligible correspond à une *activation* du processus et peut-être même élu selon sa priorité et celle du processus en cours;

- la transition éligible -> élu correspond à une *allocation* du processeur sélectionné par l'ordonnanceur. Le lancement d'un processus dépend en général de sa priorité par rapport au processus en cours d'exécution ou aux autres processus à l'état éligible;

- la transition élu -> éligible correspond à une *préemption* (ou réquisition) du processeur au profit d'un autre processus; cette préemption est décidée selon l'algorithme d'ordonnancement utilisé;

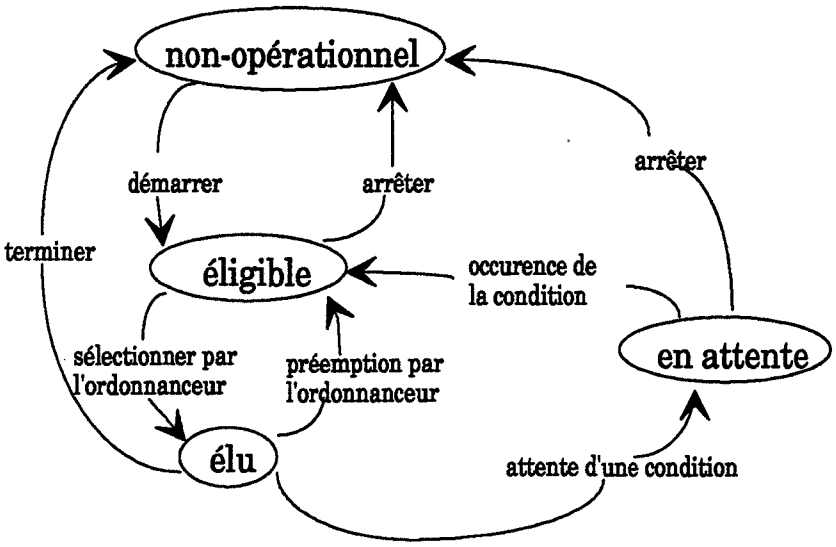


Figure 5-1 Diagramme des Transitions entre Etats d'un Processus

- la transition élu -> en attente est généralement due à un appel système ou à l'occurrence d'un événement impliquant l'*attente* d'une condition;

- la transition attente -> éligible encore appelée *réveil* du processus s'effectue suite à l'occurrence de l'événement attendu;

- la transition élu, éligible, en attente -> non-opérationnel correspond à une *terminaison* (si le processus est à l'état élu) ou à une *désactivation* (si le processus est à l'état éligible ou à l'état en attente).

Dans un système compliqué, les états d'un processus peuvent être raffinés (à condition que l'ordonnanceur les connaisse). L'état d'attente de processus peut, par exemple, être raffiné dans les trois cas suivants: attente d'une ressource, attente d'un événement ou attente durant un laps de temps (délai). La figure 5-2 montre un diagramme des changements d'état de processus plus complet. Les phases de transitions numérotées de 1 à 16 sur le diagramme sont explicitées ci-dessous:

01 Création d'un processus (new (p));
02 Destruction d'un processus (delete (p));
03 Activation d'un processus en vue de sa sélection par l'ordonnanceur;
04 Désactivation d'un processus
05 Sélection d'un processus et allocation d'un processeur par l'ordonnanceur;
06 Réquisition du processeur au profit d'un autre processus (préemption);

07 Fin d'exécution du processus en cours;
08 Blocage sur accès à une ressource non disponible;
09 Blocage sur attente de fin d'un délai;
10 Blocage sur occurrence d'un événement (généralement asynchrone);
11 Activation d'un processus forcé d'attendre l'écoulement d'un délai;
12 Activation d'un processus forcé d'attendre un événement;
13 Activation d'un processus forcé d'attendre la libération d'une ressource;
14 Activation d'un processus suite à la libération de la ressource attendue;
15 Activation d'un processus suite à l'écoulement d'un délai;
16 Activation d'un processus suite à occurrence d'un processus.

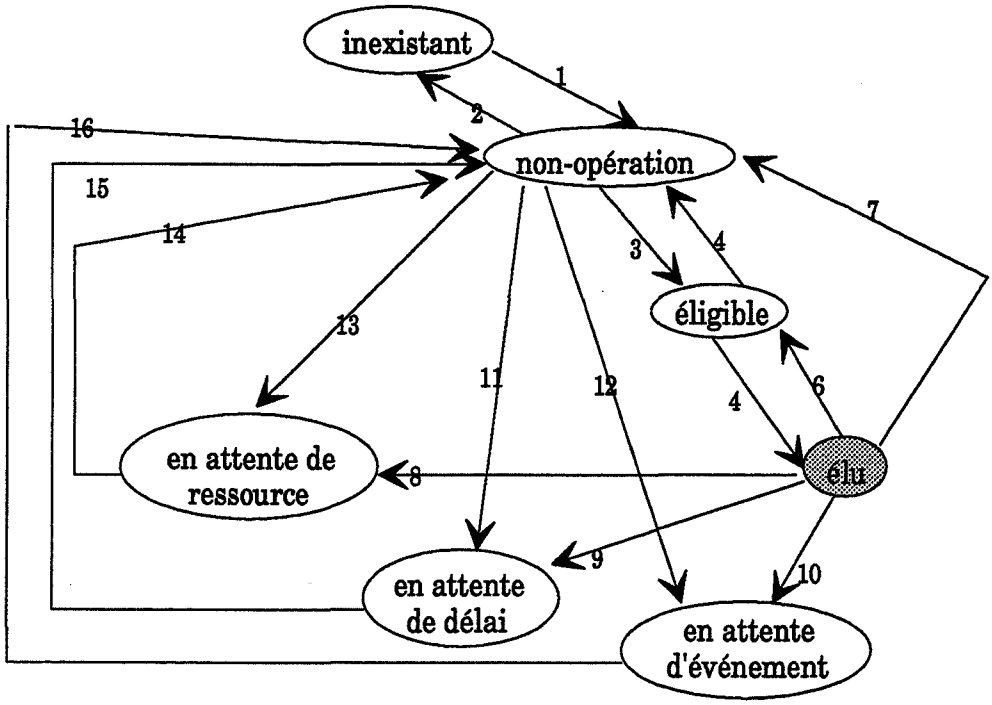


Figure 5-2 Diagramme des Etats des Processus et leurs Transitions

II.1.3 Commutation de Contexte

Les changements d'état d'un processus conduisent généralement à *une commutation de contexte* qui consiste à remplacer le contexte d'un processus P1 par celui d'un processus P2 (nouvel élu) en vue de l'exécution de ce dernier par un processeur. Un processeur est donc un agent actif responsable de l'exécution d'un ou de plusieurs processus de manière alternative ou quasi simultanée (à l'aide d'un ordonnanceur).

Il faut déjà bien distinguer la *commutation de processus* de la commutation de contexte. L'opération de commutation de contexte intègre en règle générale:

- la *sauvegarde* de l'état des registres du processus courant (fonction longjmp (env) dans le langage C, par exemple);
- la *restauration* de l'état des registres relatifs au nouveau processus (fonction setjmp ()).

Une commutation de contexte peut être effectuée suite à un appel au noyau de synchronisation (ordonnanceur) ou lors de l'occurrence d'une interruption. La commutation de processus consiste en l'abandon du traitement du processus en cours pour commencer ou continuer le traitement d'un autre processus. Une commutation de processus peut être provoquée:

- sur demande explicite du processus lui-même (mise en attente d'une condition, fin du processus);
- sur décision de l'ordonnanceur (processus en cours moins prioritaire qu'un autre processus éligible);
- par nécessité de réponse à un phénomène externe (événement).

Il en résulte qu'une commutation de processus peut éventuellement induire une commutation de contexte. Les opérations de base (fonctions "*système*") de changement de contexte d'un processus effectuées sont alors les suivantes (la figure 5-3 montre l'utilisation de ces opérations dans la méthode *suspendre()* d'un processus):

- *sauvegarde* du contexte du processus en cours (sauvegarder (processus));
- *mise* du processus en cours dans la liste des processus *en attente* (insérer (processus));
- *invocation* de l'ordonnanceur pour déterminer le processus éligible le plus prioritaire à exécuter sur le processeur (sélectionner()) dans la liste des processus éligibles;
- *restitution* du contexte du nouveau processus (restituer(processus) ou reprendre (p)).

L'appel à la méthode *suspendre*() dans le processus P1 invoque la méthode *sélectionner*() de l'ordonnanceur qui elle-même appelle la méthode *sauvegarder*() du processus invoquant. Le premier appel à sauvegarder() retourne une constante C1 (une valeur 0 en général) et sauve le contexte associé au processus P1 en vue d'une réactivation future. Suite au premier retour de la méthode sauvegarder(), le processeur est sous la direction de la méthode sélectionner() qui sélectionne, selon la figure 5-3, un nouveau processus P2.

Pour réactiver le processus P2, la méthode sélectionner() fait appel à la méthode reprendre () du processus P2 qui récupère le contexte du processus P2 en retournant une constante C2 (une valeur différente de 0) et transfère le contrôle du processeur au processus P2. Le processus P2

exécute les procédures associées, à la fin de son exécution il transfert son contrôle en exécutant sa méthode suspendre() qui déclenche le même enchaînement que précédemment.

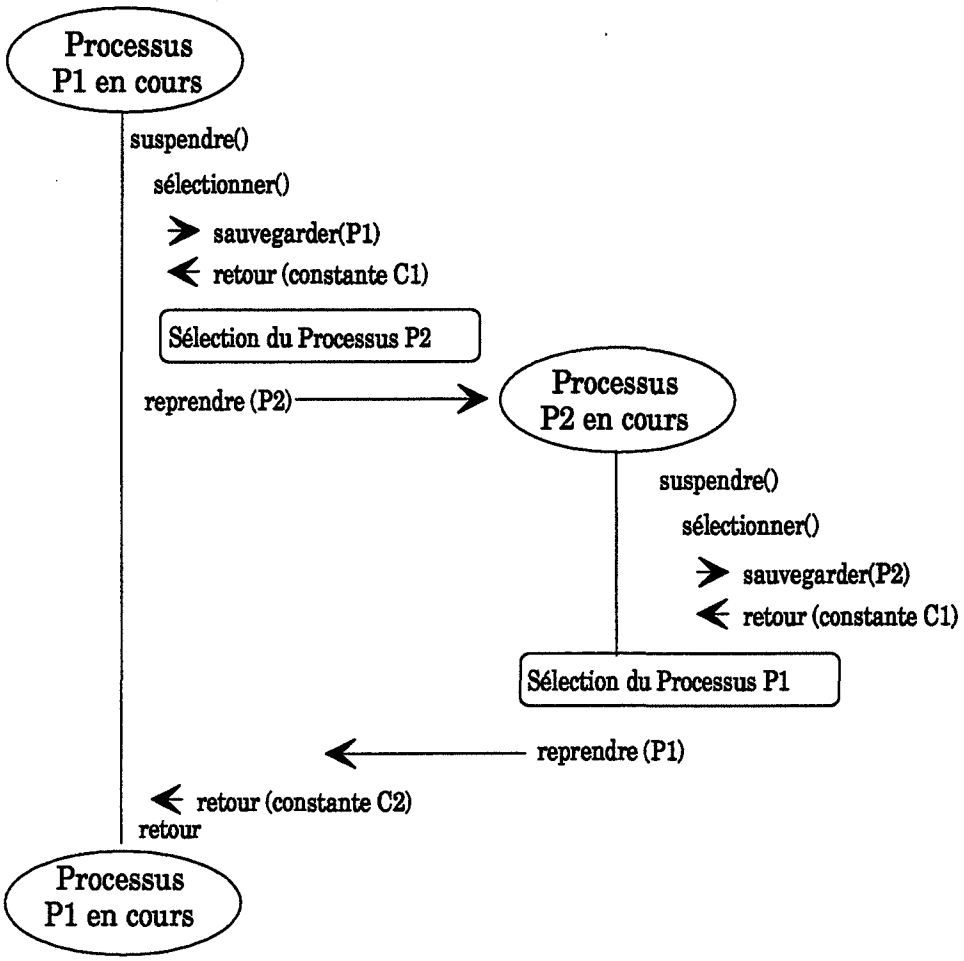


Figure 5-3 Mise en Oeuvre des Opérations dans la Méthode *Suspendre()* d'un Processus

II.1.4 Descripteur de Processus

Les structures de données manipulées par un noyau de synchronisation (ordonnanceur) sont essentiellement constituées par les descripteurs de processus. Un descripteur de processus est un objet permettant de retrouver à travers ses méthodes toute l'information propre (contexte) à un processus, à savoir:

- l'ensemble des variables propres au processus;
- la priorité et l'état du processus;
- le contenu des registres du processeur, lorsque le processus ne s'exécute pas.

L'ordonnanceur est donc une liste des descripteurs de processus. Pour retrouver toute l'information propre à un processus, il suffit de retrouver la pile du processus. La solution

générale est, soit de conserver dans le descripteur de processus le pointeur de sommet de pile du processus (figure 5-4), soit d'utiliser le mécanisme du double chaînage permettant à un descripteur de processus d'être relié par un pointeur au descripteur précédent dans la file et par un autre pointeur au descripteur suivant. Dans les logiciels de simulation, on retrouve souvent les deux solutions. Dans plusieurs ouvrages ou articles, on trouve des études sur les algorithmes de manipulation de la structure du noyau (ou précisément de la file d'attente de priorité) et des comparaisons des avantages et des inconvénients des uns par rapport aux autres: "calendar queues" [BROWN 88], "heap queues" [GONNET 76], "binomial queues ou trees" ([BROWN 78], [VUILLEMIN 78]), etc. Le choix dépend essentiellement de la complexité de l'application (nombre de processus à modéliser) et les contraintes du temps de traitement (performance du modèle).

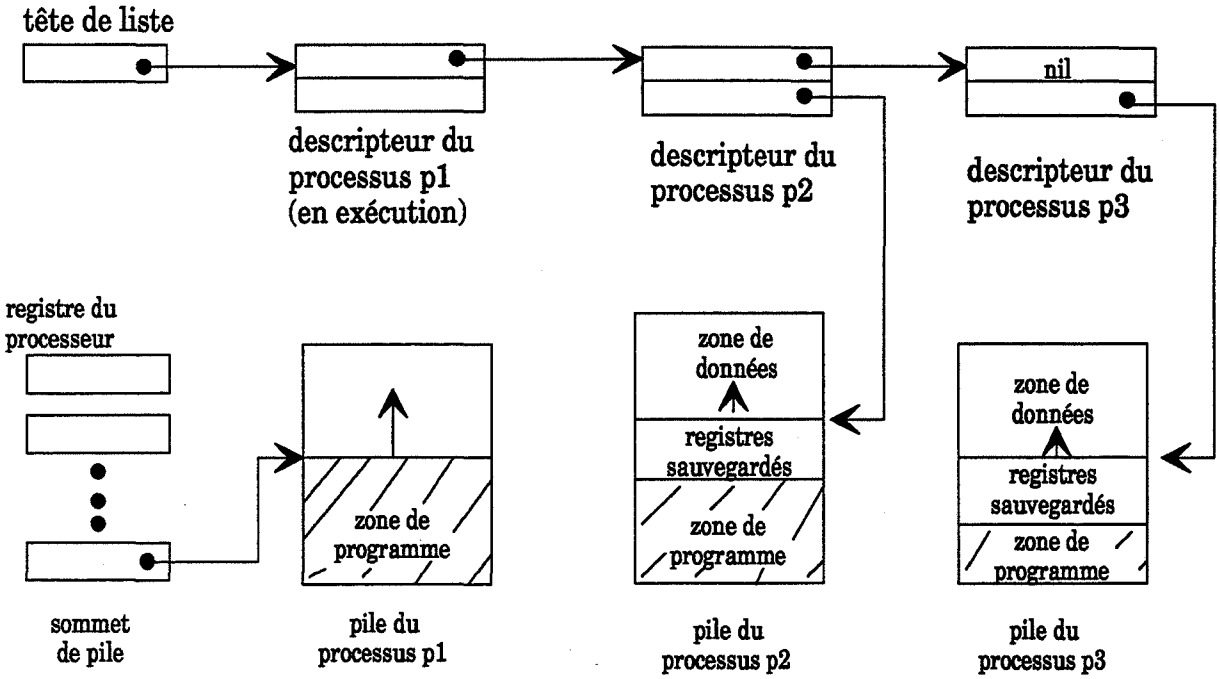


Figure 5-4 Liste des Descripteurs de Processus (le Processus p1 est en Exécution)

II.1.5 Le "Scheduler" ou l'Ordonnanceur

L'ordonnanceur est le "chef d'orchestre" d'un modèle de simulation. Il effectue le choix d'un processus parmi N **processus éligibles** et lui alloue le processeur. Le processus concerné est dit **processus élu**. L'ordonnanceur (le scheduler) a des rôles essentiels:

- garantir à chaque processus un temps d'allocation donné,
- assurer la gestion des commutations de processus,

- effectuer le choix d'un processus dans l'ensemble des processus éligibles en respectant un ordre de priorité entre processus,
- annuler de manière préemptive un processus qui monopolise le processeur,
- etc..

Les algorithmes d'ordonnanceurs classiques, implémentés auparavant dans diverses structures de files d'attente, sont conceptuellement une arborescence d'instructions, accomplies invariablement dans un ordre qui ne dépend généralement que des données initiales, et qui peut s'exprimer à partir de quelques structures de données et d'actions (procédure, bloc, séquence, boucle logique, boucle itérative, sélection, interrogative) avec un vocabulaire restreint. Ils ne font pas intervenir le temps et donnent toujours le même résultat pour des données initiales identiques [THORIN 90].

Les algorithmes d'un ordonnanceur d'un simulateur dépendent du temps, en outre, il fait intervenir:

- les objets de type **temps** (assimilable à une variable réelle ou entière à valeurs monotones croissantes, modifiée au moins à chaque accomplissement d'action d'un processus) et de type **événement** (assimilable à une variable logique qui change de valeur quand ce qu'il représente s'est produit: un objet venant d'être créé, détruit ou de prendre une certaine caractéristique, ou encore une action venant de commencer, de se terminer ou d'entrer dans une certaine phase), qui ne sont pas structurés;

- les structures d'actions de

  * **Processus:** procédure (entité) dont l'exécution est simultanée avec d'autres;

  * **Séquence Synchronisée:** suite d'ordres exécutés une fois à la survenance d'un événement, ou à chaque survenance d'un événement, ou encore seulement jusqu'à la survenance d'un événement; cela correspond dans le langage de programmation aux structures suivantes:

    lorsque (événement) (séquence),
    toutes les fois que (événement) (séquence),
    jusqu'au moment où (événement) (séquence);

  * **Boucle Synchronisée:** suite d'ordres exécutés indéfiniment jusqu'à la survenance d'un événement; cela correspond, dans le langage courant, à des ordres utilisant les

conjonctions "pendant que", "tandis que", "aussi longtemps que", "en même temps que" avec la structure suivante du langage:

aussi longtemps que (non événement) (séquence);

* **Sélection Synchronisée**: ensemble de séquences dont une au plus est exécutée selon la survenance d'un événement comme dans les structures suivantes:

selon que (événement) (séquence),
ou (événement) (séquence),
ou (événement) (séquence),
...
sinon (séquence);

* **Interrogative Temporelle**: ordres d'accès portant sur l'événement; cela correspond, dans le langage courant, à un ordre utilisant "quand" ou "depuis quand".

avec un vocabulaire plus large (attendre, déclencher, arrêter, avertir). Ces structures sont surtout composées de plusieurs arborescences accomplies simultanément et dans un ordre quelconque (simultanéité des flots d'exécution).

Un algorithme dépendant du temps ou *algorithme temps réel* est donc un ensemble de plusieurs processus en parallèle, c'est-à-dire exécutés simultanément. L'intérêt d'un algorithme temps réel, contrepartie d'une complexité accrue par rapport à ceux qui ne font pas intervenir le temps, résulte des raisons suivantes:

1) eux seuls peuvent représenter les opérations de certains systèmes en relation avec l'environnement du monde réel,
2) ils sont souvent plus naturels pour représenter des éléments autonomes d'un système, même sans relation avec l'environnement réel,
3) ils sont parfois plus efficaces, notamment en ce qui concerne les temps de réponse, et parfois plus faciles à écrire même dans le cas complexe.

On appelle *multiprogrammation* le cas où les processus se partagent une machine, y compris le processeur (il y a donc simultanéité globale et entrelacement des activités, mais pas simultanéité réelle à un instant); *multitraitement* le cas où les processus se partagent les mémoires, chacun ayant son processeur; *traitement réparti* le cas où chaque processus a sa machine et est autonome, mais relié aux autres par un réseau de communications; les combinaisons hybrides sont bien entendu possibles, en particulier le *traitement distribué* où, à

la différence du précédent, les processus ne sont pas autonomes car l'un d'eux les commande ou les contrôle, au moins à certains moments.

Nous travaillons, dans notre cas de recherche, sur une machine avec un ou plusieurs processeurs. La notion thread nous permet d'encapsuler les opérations essentielles manipulées par l'ordonnanceur (par exemple, créer, détruire, activer, désactiver (bloquer) ou terminer un processus, etc.). Diverses techniques d'ordonnancement temps réel existent: ordonnancement *circulaire*, ordonnancement *à priorité*, ordonnancement *à priorité avec des files multiples.* Dans la simulation, la technique la plus utilisée est celle à priorité. Ce type d'ordonnancement permet de réaliser des noyaux de synchronisation totalement indépendants du matériel (pour un processeur donné). Les noyaux sont compacts et efficaces. Leur emploi est cependant limité aux applications comprenant des processus qui ont peu de calcul et pas beaucoup d'entrées/sorties. C'est pour cela que pour la plupart des ordonnanceurs pour la simulation on a choisi l'ordonnancement à priorité.

II.2 Relations entre Processus

Il est très rare qu'un processus (une entité) soit isolé des autres. Plusieurs processus s'exécutent en général d'une manière enchevêtrée et/ou simultanée dans un ordre a priori indéterminé. Cela implique l'existence des relations de dépendance entre eux (même si les processus ne se connaissent pas entre eux: il n'y a pas de lien direct entre processus), puisque l'un peut a priori intervenir sans le savoir pendant le laps de temps où un autre a temporairement suspendu le respect des contraintes, donc trouver et/ou rendre un état incohérent.

Les processus ne sont indépendants que si, et seulement si:

- ils n'ont pas d'environnement commun (ni ressource, ni activité extérieure connue de plusieurs processus à la fois), pour que chacun soit seul responsable du respect des contraintes,
- et ils ne se connaissent pas non plus entre eux, pour que ni leurs ressources ni leurs activités intérieures ne puissent être influencées directement ou non, par aucun autre.

Ce qui revient à dire qu'ils sont tous isolés. Ce n'est pas le cas dans la réalité, surtout dans le domaine de la production. Dans tous les autres cas, une relation de dépendance existe entre les processus (même si elle ne se manifeste pas à chaque exécution):

- qu'elle soit volontaire parce que les processus concourent à une même fonction (**coopération** pure), par exemple, deux machines en cascade pour traiter une pièce, sont

en coopération et en dépendance évidentes, sans qu'il soit nécessaire que l'une des machines connaisse l'état de l'autre.

- ou involontaire du fait des limitations de l'environnement (**compétition** pure), par exemple, plusieurs processus voulant acquérir la même ressource partagée à un instant sont en compétition pure.

On peut distinguer deux types d'influence entre processus: une influence directe et une influence indirecte. Une **influence directe** ne peut provenir que d'un ou plusieurs processus (mais non de ressources ni d'activités) et s'exécute sur: 1) un ou plusieurs processus (l'arrêt de processus par un autre), 2) une ou plusieurs ressources par modification d'état (l'utilisation d'une ressource partagée avec un autre processus), 3) une ou plusieurs activités par conditionnement logique ou temporel. Des influences directes sur un ou plusieurs processus peuvent se combiner en cascade pour constituer une **influence indirecte**.

Thorin [THORIN 90] a distingué neuf cas intéressants d'influences entre les processus, les ressources et les activités, c'est-à-dire de neuf types de relations de dépendance.

1). Un processus agit globalement sur des processus;
2). Un processus agit globalement sur des ressources;
3). Un processus agit globalement sur plusieurs activités;
4). Plusieurs processus agissent successivement sur un processus;
5). Plusieurs processus agissent successivement sur une ressource;
6). Plusieurs processus agissent successivement sur une activité;
7). Plusieurs processus agissent successivement sur des processus;
8). Plusieurs processus agissent successivement sur des ressources;
9). Plusieurs processus agissent successivement sur des activités

Les trois premiers cas d'influence ont en commun la caractéristique d'être globaux. La difficulté majeure de ces relations vient du fait qu'a priori d'autres processus pourraient agir (bien que chacun soit ici vu un à un) et rendre le tout incohérent, en se considérant chacun comme seul. Il faut donc que le processus puisse opérer par des actions primitives atomiques (des services rendus par le processus). Cela dépend du temps exécutif (modèle de processus) utilisé, du langage de programmation choisi, etc.

II.3 Communication et Synchronisation entre Processus

Les processus coopèrent en vue de la réalisation des activités communes. On distingue deux sortes de coopération, la coopération *temporelle* faisant intervenir les notions de blocage et de

déblocage de processus, la coopération *spatiale* se rapportant à l'échange d'informations entre processus. Ces deux types de coopération caractérisent respectivement la synchronisation et la communication entre processus.

II.3.1 Communication Interprocessus

Les processus ont besoin d'échanger des informations pour coopérer à l'exécution d'une application. Les formes de communication utilisées en monoprogrammation telles que les paramètres de procédure ou les résultats de fonction ne sont pas adaptées au modèle d'exécution pseudo-simultané des processus. La seule forme de communication possible semble être la communication par zone de données commune ou par des messages. Parce qu'il autorise a priori des échanges réciproques simultanés, ce mode de communication conduit à un type de solution où tous les échanges se font à l'aide de mécanismes spéciaux sous le contrôle du système. Ces mécanismes revêtent alors des formes multiples: messages, tubes, boîtes à lettres, signaux, sémaphores, événements, etc. Cette multiplicité s'explique par le fait qu'aucun mécanisme n'est vraiment satisfaisant et ne permet pas de couvrir l'ensemble des besoins de communication. Chacun est adapté à un modèle d'échange particulier. Nous présentons ici deux modes les plus utilisés dans la simulation.

II.3.1.1 Communication par zone de données commune

La façon la plus simple et la plus rapide de transmettre des données est de ne pas en transmettre. Il suffit pour cela de disposer d'une zone de mémoire commune dans laquelle seront déposées les données en libre accès. L'allocation de cette zone est réalisée à l'aide de variables globales (ou statiques en langages C++) ou à l'aide d'une requête spécifique.

Il n'y a pas de problèmes d'*interblocage* et d'*exclusion mutuelle* dans le partage de variables (simples) globales ou statiques. Le concept d'encapsulation renforce encore la sécurité d'exclusion mutuelle [BUHR et al. 92]. Cela dépend de la conception des classes d'objets (attributs statiques, publiques, protégés ou privés) et de l'application (classes abstraites, classes virtuelles, variables statiques).

Les problèmes posés sont liés plutôt à l'accès à des objets complexes et partagés. Dans un système monotâche, l'accès aux zones de données communes est séquentiel. L'ordre dans lequel il s'effectue est imposé par la structure du programme. Dans un système multitâche, l'accès aux objets partagés est fonction de l'ordonnancement des processus. Il existe donc des *accès concurrents* où plusieurs processus lisent ou écrivent des données partagées, et où le résultat est fonction de l'ordonnancement des processus. La fiabilisation des programmes comportant des accès concurrents est donc difficile voire impossible.

Pour éliminer les accès concurrents, il suffit de trouver le moyen d'interdire l'accès simultané à une donnée (un objet) partagée. Ce moyen s'appelle l'*exclusion mutuelle*. L'exclusion mutuelle empêche d'accéder à un objet partagé ou à un attribut d'un objet si celui-ci est utilisé par un autre processus. La partie du programme qui conduit à un conflit d'accès est appelée *section critique*. Le problème de l'accès concurrent est résolu si un seul processus est autorisé à accéder à une section critique.

Différentes techniques existent pour assurer l'exclusion mutuelle ([BUHR et al. 92], [DORSEUIL 90], [SCHIPER 86]): exclusion mutuelle par masquage des interruptions, exclusion mutuelle par variable *verrou* et attente active, exclusion mutuelle par *sémaphore*, exclusion mutuelle à l'aide de la procédure *moniteur*, exclusion mutuelle à l'aide d'un gestionnaire de ressources, etc.

II.3.1.2 Communication par messages (modèle producteur/consommateur)

La communication par zone des données commune nécessite de gérer les échanges dans les moindres détails. La fréquence d'utilisation des modèles de communication par message (modèle producteur/consommateur ou modèle boîtes aux lettres) incite à réaliser des mécanismes d'échanges d'informations de plus haut niveau. Ces mécanismes doivent permettre l'échange d'informations entre processus et même entre machines distantes en évitant au programmeur les détails de l'exclusion mutuelle et de la synchronisation par les messages (données). Ces procédés s'appellent l'*échange par message*.

Le modèle producteur/consommateur est une généralisation de l'échange d'informations à travers une zone de mémoire commune aux deux processus (figure 5-5). La communication se passe selon le schéma suivant:

- le producteur produit des messages et les dépose dans une zone de mémoire commune (une file d'attente des messages en général) aux deux processus;

- le consommateur prélève les messages et les utilise;

- les messages, dans la zone de mémoire commune, sont gérés par la file d'attente et par un événement "File-non-Vide" généré par le producteur si la file d'attente est vide ou un événement "File-non-Full" est généré par le consommateur si la file est pleine.

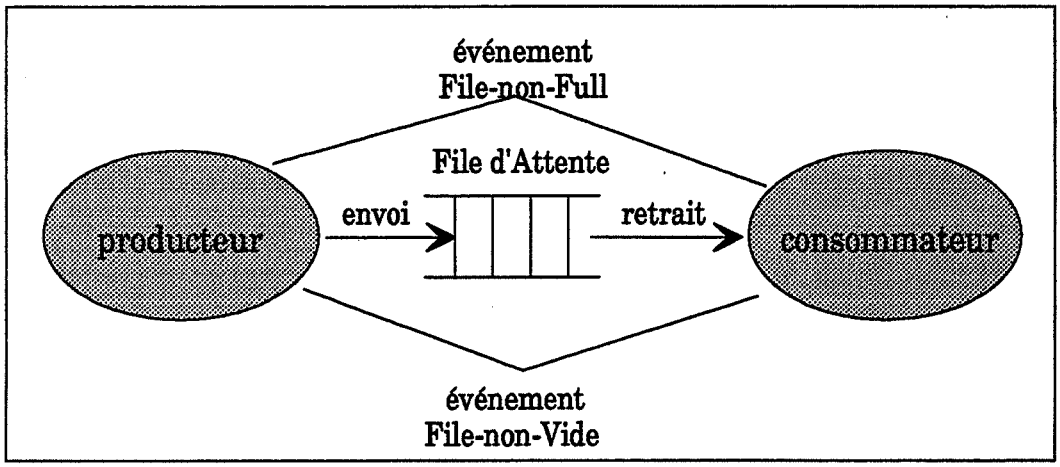


Figure 5-5 Modèle Producteur-Consommateur

Les opérations (algorithmes) essentielles d'une telle structure peuvent être schématisées dans la figure 5-6. L'avantage de ce modèle est l'indépendance réciproque de deux processus. Plus la capacité de la file d'attente est grande, plus elle permet d'absorber des différences temporaires de débit entre le producteur et le consommateur. Un autre avantage est que cette file d'attente nous permet de définir différentes stratégies de gestion des messages (gestion des flux d'information ou gestion de transactions).

Algorithme du Producteur

Répéter
  Si PLEINE (File) Alors
    ATTENDRE (File-non-Full)
  Finsi

  ENVOYER (message, file)
  SIGNALER (File-non-Vide, Consommateur)

A l'infini

Algorithme du Consommateur

Répéter
  Si VIDE (File) Alors
    ATTENDRE (File-non-Vide)
  Finsi

  RETIRER (message, file)
  SIGNALER(File-non-Full, Producteur)

A l'infini

Figure 5-6 Algorithmes du Modèle Producteur-Consommateur

II.3.2 Synchronisation Interprocessus

Les processus d'une application n'évoluent pas de façon indépendante. Nous avons identifié neuf relations temporelles entre les processus. On appelle aussi ces relations des *synchronisations*. La synchronisation consiste à imposer un ordre sur l'exécution des instructions des processus. Ce sont des mécanismes de communication où l'information échangée est utilisée dans le but de synchroniser les processus. On peut dire que l'exclusion

mutuelle peut être vue comme un cas particulier de synchronisation: un processus ne peut entrer en section critique qu'après la sortie du processus précédent. La synchronisation de plusieurs processus entre eux nécessite la construction d'un mécanisme permettant à un processus:

- d'activer un autre processus par un signal en lui transmettant éventuellement de l'information. Les activations peuvent être mémorisées ou non. Dans le cas des activations mémorisées, le signal peut être reçu à tout moment (à condition que le processus récepteur soit créé) et sera pris en compte lors du retour du processus récepteur à l'état élu ou éligible. Dans le cas des activations non mémorisées, le signal est perdu si le processus ne l'attend pas. Différentes exceptions sont donc nécessaires à implémenter pour traiter ces phénomènes non prévus;

- de se bloquer lui-même ou d'en bloquer un autre par un message d'événement;

- d'envoyer un signal de réveil vers un ou plusieurs processus. Selon que ce signal soit mémorisé ou non par le processus récepteur, le processus émetteur ne se bloque pas ou, au contraire, se bloque en attendant que le processus récepteur renvoie un message de retour.

Le mécanisme de synchronisation est **direct** lorsque le processus envoie un signal en désignant directement le récepteur ou lui-même; il est **indirect** lorsque le processus agit par l'intermédiaire d'un mécanisme qui lui seul connaît les processus concernés par l'action désirée.

Dans la synchronisation directe, trois types de méthodes du processus permettent d'implémenter ces mécanismes:

- La méthode *bloquer (processus)* ou *interrompre (processus)* envoie un signal permettant de suspendre un processus désigné ou lui-même si on ne précise pas le récepteur;
- La méthode *réveiller (processus)* ou *reprendre (processus)* envoie un signal vers le processus désigné;
- La méthode *suspendre(), dormir ()* ou *terminer ()* suspend le processus y faisant appel.

Ces trois types de méthodes, en incluant les deux méthodes de changement de contexte constituent le minimum de cinq primitives de la gestion des processus et elles sont implémentées dans un objet abstrait **Processus**. La plupart des outils de simulation à objets fournissent ces mécanismes à travers soit des primitives du langage, soit des classes d'objets spécifiques.

Dans la synchronisation indirecte, les noms des processus à synchroniser ne sont pas directement désignés par l'opération. La synchronisation s'effectue par l'intermédiaire d'objets communs à travers des primitives assurant l'exclusion mutuelle de ces objets ([SCHIPER 86], [GEHANI 88], [BUHR et al. 92]). Deux objets sont généralement utilisés, les **sémaphores** et les **variables d'événements**. Un troisième objet, appelé variable de **rendez-vous**, apporte une contribution originale en permettant une généralisation de ce moyen de synchronisation à plusieurs processus.

III Outils et Langages Orientés Processus pour la Simulation à Objets

Actuellement, les outils et langages à objets pour la simulation se multiplient. Il est difficile de faire une synthèse de ces outils. Notre objectif de recherche est de construire une plate-forme pour la simulation orientée-objets; il est donc naturel de construire notre outil autour d'un langage plate-forme C++. On peut classifier, cependant, les outils de simulation que l'on rencontre dans la littérature selon trois catégories:

- langages orientés-objets avec des primitives de supports (ou constructions syntaxiques) pour la simulation comme Smalltalk [GOLDBERG et al. 83], Simula [BIRTWISTLE et al. 73], ADA [BARNES 88], MODSIM II [BELANGER 90b], Extend [HILL 93], etc.;

- extensions des langages en incluant des primitives de communication et de synchronisation entre processus pour la simulation comme Concurrent C++ [GEHANI et al. 88], SIM++ [LOMOW et al. 90], SIM Plus Plus [ROSE 92], DEVS/Scheme [ZEIGLER 89], etc.;

- construction des librairies des classes spécifiques pour l'ordonnancement des classes d'objets "threads" ou des événements comme la librairie des classes tâches de AT&T ([STROUSTRUP et al. 87], [SHOPIRO 87]), Simple++, Interactor [LABRECHE 90], Meijin++ [NICOLAS 93], PRESTO [BERSHAD et al. 88], etc.

III.1 Langages avec des Primitives de Supports pour la Simulation

DE VETTOR [DE VETTOR 91] a fait une comparaison de différents langages qui fournissent des primitives pour la simulation tels que Smalltalk, ADA, AIR1C, C++, Actor, etc. Puisque nous avons commencé nos recherches à partir du langage MODSIM II (un premier langage de simulation à objets), nous présentons donc les caractéristiques principales de ce langage et expliquons pourquoi nous l'avons abandonné.

Le langage modulaire de la simulation MODSIM II est un langage universel et modulaire, qui fournit un support pour la programmation orientée-objets et la simulation à événements discrets. Ses syntaxes et structures sont très similaires à celles des langages Modula 2 et ADA. L'approche de la simulation adoptée est l'approche par processus comme dans le langage de simulation SIMSCRIPT II.5. Les concepts d'encapsulation, d'héritage simple et multiple, les mécanismes polymorphisme et liaison dynamique sont aussi supportés par ce langage.

Les classes d'objets sont spécifiées et implémentées dans deux modules correspondants de la librairie: les **modules de définition** qui définissent les attributs et les méthodes d'objets (dans ces modules, il n'y a pas de code d'exécution), et les **modules d'implémentation** qui contiennent le code qui implémente toutes les procédures et les méthodes d'objets. Les objets, les procédures, ou les variables mentionnés dans les modules de définition peuvent être importés par les autres modules. Par contre, les variables ou procédures déclarées dans le module d'implémentation ne sont pas accessibles par les autres modules. Tout module peut être compilé séparément. Un programme de MODSIM II est composé donc de trois types de module: un *module principal* et plusieurs modules de définition et d'implémentation.

La simulation est supportée directement par des "built-in" primitives. Il fournit ainsi une librairie des classes d'objets prêtes à être utilisées. Pour définir une méthode synchrone, par exemple, on peut utiliser la méthode suivante:

ASK objet **TO** méthode (liste des paramètres)

Une méthode asynchrone est définie comme suit:

TELL objet **TO** méthode (liste des paramètres) <**IN** temps>

Pour définir une méthode asynchrone avec l'objet TriggerObj afin de synchroniser deux objets travaillant en parallèle, on utilise la primitive WAIT:

WAIT triggerobject **TO File**
TELL triggerobject **TO File**

Pour interrompre une méthode asynchrone, on peut utiliser les primitives **Interrupt** (objet, nom de méthode), **InterruptMethod ACTID** ou **Terminate**. Les classes d'objets pré-construites avec ces primitives dans la librairie de MODSIM II pour la simulation

comprennent: différentes files d'attente (QueueObj, StackObj, RandObj), l'objet ressource (ResourceObj), les objets statistiques (StatQueueObj, SINTEGER, SREAL, ...), etc.

Le mécanisme d'ordonnancement peut être structuré comme indiqué dans le schéma suivant (figure 5-7): une liste des objets actifs, et pour chaque objet actif, on garde une liste de ses activités parallèles.

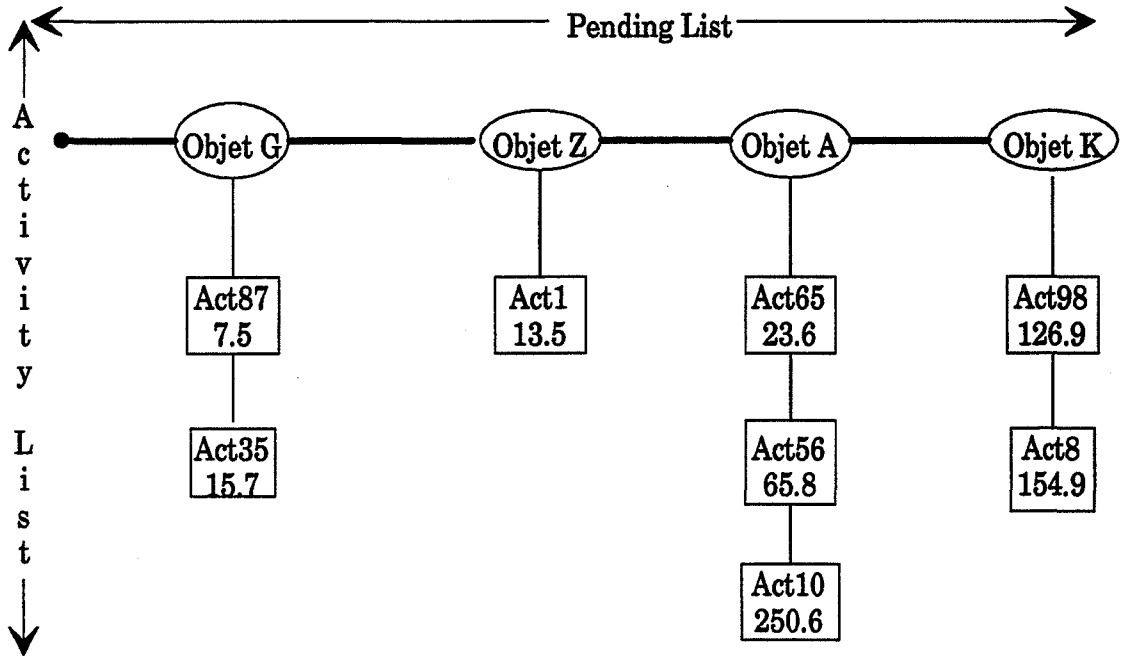


Figure 5-7 Structure des Files d'Attente d'Objets Actifs et ses Activités

Malgré ses qualités, ce langage n'est pas retenu pour le développement de nos applications pour les raisons suivantes:

1) La visibilité de controle de temps du système est cachée de l'utilisateur. Nous ne faisont intervenir que l'horloge du système à travers quelque primitives prédifinies. Il est donc difficile de construire des classes d'objets autonomes.

2) Nous voulons travailler sur une plate-forme de simulation sur micro-ordinateur. Bien qu'il existe des logiciels graphiques (GRAPHICS) pour MODSIM II sur des stations, ils restent inutilisables sur des PCs.

3) MODSIM II nous semble difficile à adapter pour des projets complexes ([YE 93], [ROSE 92]). La compilation est très lourde et le temps de développement assez long.

Puisque le langage de programmation C++ est devenu le standard pour le développement d'une plate-forme tant au niveau du modèle qu'au niveau d'interface, nous avons décidé de développer notre simulateur en utilisant ce langage dès le début de l'année 1992. Nous avons étudié plusieurs bibliographies avant de prendre cette voie ([SHOPIRO 87], [STROUSTRUP et al. 87], [GEHANI et al. 88], [BERSHAD et al. 88], [LABRECHE 90], [BUHR et al. 90], etc.). Le premier logiciel acquis de l'extension du C++ pour la simulation est une librairie scientifique avec des objets threads: Meijin++ [NICOLAS 91]. Par la suite, nous avons obtenu une extension du langage C++ pour la simulation: SIM Plus Plus [ROSE 92].

III.2 Extensions des Langages pour la Simulation

Le progiciel SIM Plus Plus (SIM_P_P), une bibliothèque de classes d'objets avec des primitives pour la simulation, est un langage de simulation à événements discrets orienté processus, basé sur le langage de programmation C++. Il a été développé à l'"Institute of Telematics" à l'Université de Karlsruhe en Allemagne.

Le choix de l'approche par processus a été fait dans l'optique de simplifier la conception de modèles complexes, conception qui n'est pas toujours évidente avec la simulation à événements discrets. Pour cela, le développement de SIM_P_P a été fortement influencé par le langage MODSIM II. La différence entre ces deux langages est que SIM_P_P rend explicite la définition du processus comme un objet: un objet coroutine (processus léger) qui permet d'implémenter facilement des objets ayant un ou plusieurs processus.

Un objet dans SIM_P_P est donc composé de trois types d'éléments:

- "**Data elements**" qui définissent les caractéristiques (attributs et relations) de la classe d'objet. Ils peuvent être soit publiques, soit privés.
- "**Methods**", ce sont des procédures identiques à celles écrites en langage C++, mais elles ont le droit d'accéder aux "data elements" des classes d'objets associées.
- "**Process**", ce sont des méthodes qui peuvent influencer sur le temps de simulation en interrompant leur exécution et en étant réactivées après un laps de temps.

Les processus dans SIM_P_P doivent être déclarés dans une classe d'objet comme suit:

```
class <class name> {
  Process (Proc name) (<parameters>);
}
```

Puis dans une implémentation de la classe, la définition du processus se fait comme suit:

```
PROC (<class name>, <proc name>) (<parameters>)
{ (<local parameters>);
   StartProcess (<process name>);
        /* the code */ ...
   EndProcess();
}
```

Les processus doivent être activés pour qu'ils puissent être "scheduled" et "maintained" par l'ordonnanceur proprement, pour cela il y a deux manières:

- Activation asynchrone d'un processus: le demandeur (l'objet qui active ce processus) n'attend pas que le processus en cours soit terminé pour que sa requête soit satisfaite:

  Activate (<proc name>, (<parameters>), double **time** = n);

  le processus <proc name> sera déclenché immédiatement au temps T = SimTime + n.

- Activation synchrone d'un processus: le demandeur est obligé d'attendre que le processus <proc name> soit terminé avant de poursuivre son exécution et le processus en cours verra son exécution interrompue jusqu'à la fin du processus <proc name>.

  BOOLEAN **WaitFor** (<proc name>, <parameters>);

L'écoulement du temps de simulation dans un processus peut se faire avec le primitive suivant:

BOOLEAN **Wait** (double **time**);

Ce processus est interrompu pendant une durée égale à "**time**". L'arrêt d'un processus se fait par l'appel à la routine **Terminate** (). Tous les processus activés de manière synchrone et concernés par ce processus seront terminés également.

On peut interrompre un processus. Pour cela il faut utiliser les primitives:

Interrupt (<obj>, <obj class>, <proc>) pour interrompre un processus d'un objet;

ou

InterruptAll (<obj>) pour interrompre tous les processus d'un objet.

D'autres méthodes définies dans SIM_P_P qui permettent la gestion des processus sont:

- **CreateProcess**(): Créer un nouveau processus et sauvegarder son contexte;
- **EndProc**(): La fin d'un processus (arriver à son terme);
- **Cleanup**(): Détruire les processus qui sont dans la liste de l'ordonnanceur (à la fin de la simulation);
- **MainTask**(): C'est la première tâche à effectuer lors d'une simulation, elle déclenche l'exécution des processus;
- **StartSimulation**(): Précéder *Maintask* dans la séquence de démarrage d'une exécution de la simulation;
- **InsertEnv**(): Ordonnancer les processus en les insérant dans la liste des processus d'attente "WaitList";
- **RemoveProc**(): Détruire un processus;
- **NextTask**(): Déterminer le prochain processus à exécuter.

Dans la bibliothèque SIM_P_P, trois classes d'objets sont prédéfinies pour gérer la synchronisation des processus: une classe "**TriggerObj**" identique dans le langage MODSIM II, une classe "**Rendez-VousObj**" permettant d'avoir un mécanisme du type: un processus ne sera déclenché que quand tous les participants (invités au rendez-vous) seront présents, et une classe "**Semaphore**" servant à gérer les synchronisations au niveau des ressources partagées.

Les autres utilitaires pour la simulation tels que les différentes files d'attente, les générateurs de nombres aléatoires, les classes statistiques et la représentation graphique des résultats sont ainsi définis. Le langage SIM_P_P appartient à la plate-forme du développement: son architecture est similaire à celle que nous avons présentée dans le chapitre III (figure 3-3). Une thèse basée sur cet outil est en cours dans le département [OUZOUT et al. 94].

Comme la définition des objets autonomes n'est pas dans la philosophie de base de ce langage, il est difficile d'implémenter les concepts d'objets autonomes (agents) du chapitre IV. Notons qu'il y a des différences entre la notion d'objet actif et d'objet autonome ou contrôlé. Un objet actif fait avancer le temps de simulation, et un objet autonome (contrôle) a non seulement les caractéristiques d'un objet actif, mais également son propre flot de contrôle. Le concept thread permet d'avoir plusieurs flots de contrôle (c'est-à-dire qu'il a ses propres activités et ressources (attributs)) dans le même espace d'adressage. Il est naturellement intéressant d'implémenter la simulation avec des outils qui supportent le concept thread.

III.3 Librairies des Classes "Threads" pour la Simulation

Plusieurs librairies des classes basées sur le langage C++ et le concept thread (mémoire partagée et objets communiquants) existent: la librairie des classes basées sur la notion tâche

du laboratoire AT&T Bell, la librairie des classes *PRESTO* basée sur la notion thread pour écrire des systèmes en temps réel, l'exécutive du temps réel *Interactors* basé sur la notion thread, la librairie scientifique de *Meijin++*, etc. Ils reprennent tous la notion de co-routine que l'on trouve dans le langage C ou C++ ([BINDING 85], [SHOPIRO 87], [GEHANI et al. 88], [AKERBAEK 93]) et la enrichissent pour supporter le concept thread.. Une forte tendance de concept multi-threads existe même dans le futur système d'exploitation ([VARHOL 94], [POUTAIN 94], [WAYNER 94]). Nous présentons l'une de ces librairies à travers la librairie Meijin++ que nous avons exploité dans notre cas de recherche.

Le progiciel Meijin++ est une bibliothèque scientifique de composants prêts à être utilisés. Il transforme le compilateur C++ en un outil puissant de modélisation et de simulation. Meijin++ représente un système réel par deux types d'objets: objets actifs et objets passifs. Un *objet actif* a une activité autonome, il est implémenté comme un processus "léger" (light-weight process) et il interagit avec l'extérieur à travers des messages. On l'appelle parfois "objet autonome" si on ne précise par leur flot de contrôle interne. Dans un modèle de simulation, l'objet actif est manipulé et maintenu par le noyau de synchronisation (**Scheduler** dans Meijin++) à travers ses méthodes (fonctions membres). Un *objet passif* n'est pas géré directement par le **Scheduler**, il est crée, manipulé et détruit par des objets actifs.

Dans Meijin++, tous les objets actifs dérivent d'un objet actif abstrait **Thread** qui est implémenté comme un processus "léger" ou précisément une coroutine dans la mémoire. La structure d'objet *Thread* est la suivante (figure 5-8): elle inclue les attributs locaux comme l'espace d'exécution *buffer*, les attributs statiques (globaux) comme l'horloge *clock*, le pointeur sur l'objet échéancier *scheduler*, le pointeur sur le processus actif courant *current*, l'attribut de l'adresse courante d'exécution *basePtr*, l'attribut de l'état de la machine *env*, etc., et les trois méthodes: **suspendre** (*switch_out*), **reprendre** (*switch_back*), **commencer** (*spawn*). La méthode virtuelle *run*() du Thread va être surchargée et raffinée dans les classes d'objets dérivées pour définir leurs propres comportements temporels (méthode *fonctionner*).

En utilisant cet objet abstrait, on dérive un objet processus **Process** qui ajoute des méthodes membres pour synchroniser les activités des objets actifs et résoudre d'éventuels conflits entre deux objets dont les activités se recouvrent dans le temps. En fait, chaque processus peut être décomposé en plusieurs "segments" qui sont exécutés séquentiellement ou à différentes étapes. Un objet processus peut être en différents états. Il peut être en:

- état élu ou en cours (**CONTROL**),
- état éligible ou prêt (**SCHEDULED**),
- état bloqué ou en attente (**IDLE**),

- ou état mort ou non-opérationnel (**KILLED**).

La communication entre les différents objets actifs dans le Meijin++ se base sur les trois différents scénarios suivants:

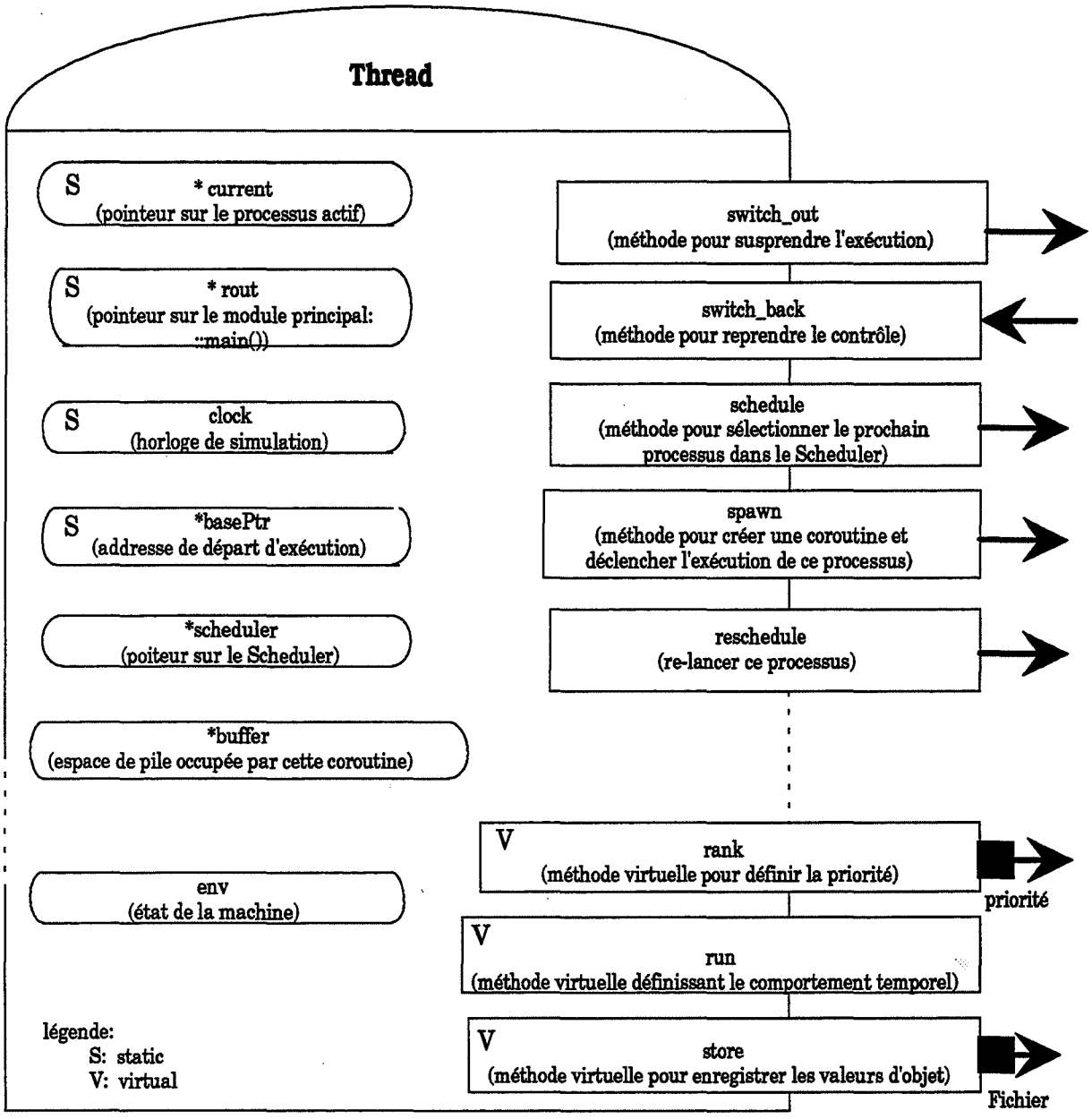


Figure 5-8 Diagramme de Structure de l'Objet Thread

- Exclusion mutuelle pour le partage des ressources. Une ressource ne peut être utilisée que par un processus à la fois et elle sera libérée quand le processus se termine.

- Communication par des messages asynchrones avec une file d'attente commune. Quand l'émetteur (producteur) envoie une entité (un message asynchrone) dans la file d'attente en aval, il va vérifier cette queue: si elle est vide, il va générer un message "File-non-Vide" pour débloquer le processus en aval; si elle est pleine, il se bloque lui-même; sinon, il dépose l'entité dans la file et continue son activité. Quand le récepteur (consommateur) prend une entité dans la file d'attente en amont, il va aussi vérifier cette queue: si elle est pleine, il va générer un message "File-non-Full" pour débloquer le processus interrompu en amont; si elle est vide, il va se bloquer; sinon, il enlève l'entité de la file et poursuit son activité. On le désigne aussi comme mode producteur-consommateur (communication bloquante).

- Coopération directe entre les processus par des messages synchrones. Un processus (client) se bloque par un autre processus (serveur) et attend d'être réveillé par son serveur. On l'appelle souvent le mode client-serveur.

On enrichit donc les méthodes membres suivantes concernant la synchronisation, la communication et la gestion du temps de simulation dans l'objet Process:

* Avancer l'horloge du système par une période fixe du temps en exécutant:

  Process::delay(unsigned);

* Etre bloqué sans condition jusqu'à la réactivation par un autre processus en exécutant:

  Process::block(Thread*);

* Etre bloqué jusqu'à la terminaison d'un processus ou d'un ensemble de processus en exécutant:

  Process::pending_one(Thread, unsigned)**
  ou
  Process::pending_all(Thread, unsigned);**

* Bloquer le processus actif courant jusqu'à ce qu'il se termine en exécutant:

  Process::pre_empty();

En combinant ces méthodes, on obtient le diagramme des transitions d'état d'un objet actif suivant (figure 5-9):

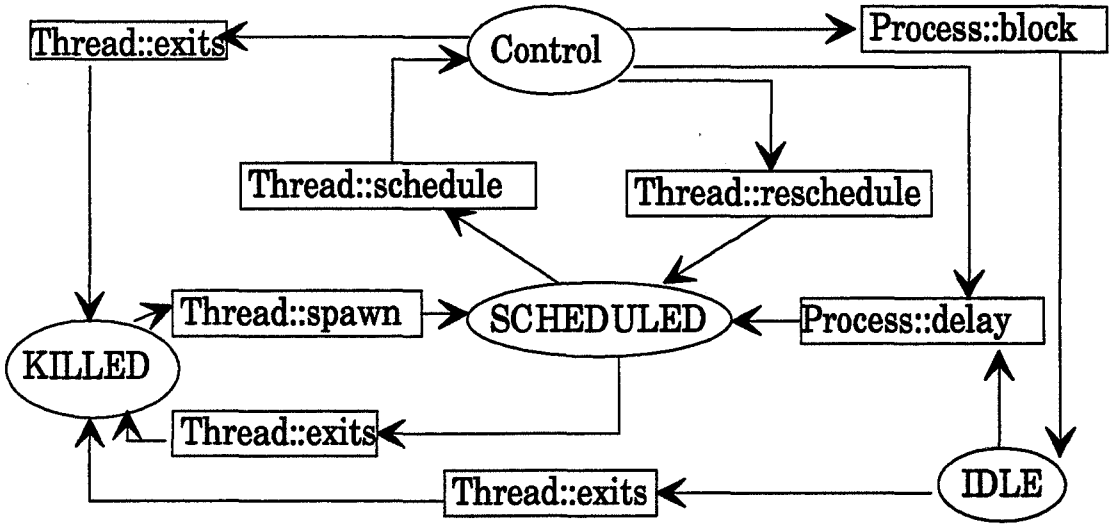


Figure 5-9 Diagramme des Transitions d'Etat d'Objets Actifs

A partir de ces deux classes d'objets de base, Meijin++ fournit trois classes d'objets actifs pour définir des objets autonomes de la simulation: une classe **Server** qui sert à définir un objet actif qui a une file d'attente en entrée et une file d'attente en sortie, une classe **QNode** qui a n files d'attente en entrée et n files d'attente en sortie et une classe **Timer** qui sert à définir des méthodes qui vont arrêter l'objet du type *Server* ou *QNode* (un processus de panne, par exemple) pendant un laps de temps.

Les autres classes d'objets tels que les classes d'objets utilitaires/auxiliaires pour la simulation comme les générateurs des nombres aléatoires, les classes d'objets analyses et statistiques, les classes d'objets d'application comme les différents ordonnanceurs (Heap, CalQueue, SortList, ...), les classes d'objets mathématiques, etc., sont ainsi prédéfinies dans la bibliothèque (figure 5-10). Meijin++ fournit aussi une bibliothèque des classes d'objets d'interface graphique appelé *Plots Video* pour le MS-DOS.

Dans la suite, nous exploitons l'implémentation des objets développés dans les deux chapitres précédents à travers les classes d'objets du Meijin++: comment définir les services des objets actifs sémantiques identifiés dans le chapitre III à partir des méthodes d'objets *Thread* et *Process*. La hiérarchie des classes d'objets passives (classes d'objets transactionnels et classes d'objets décisionnels) basée sur la classe d'objet **Item** de Meijin++ est présentée à la fin du chapitre.

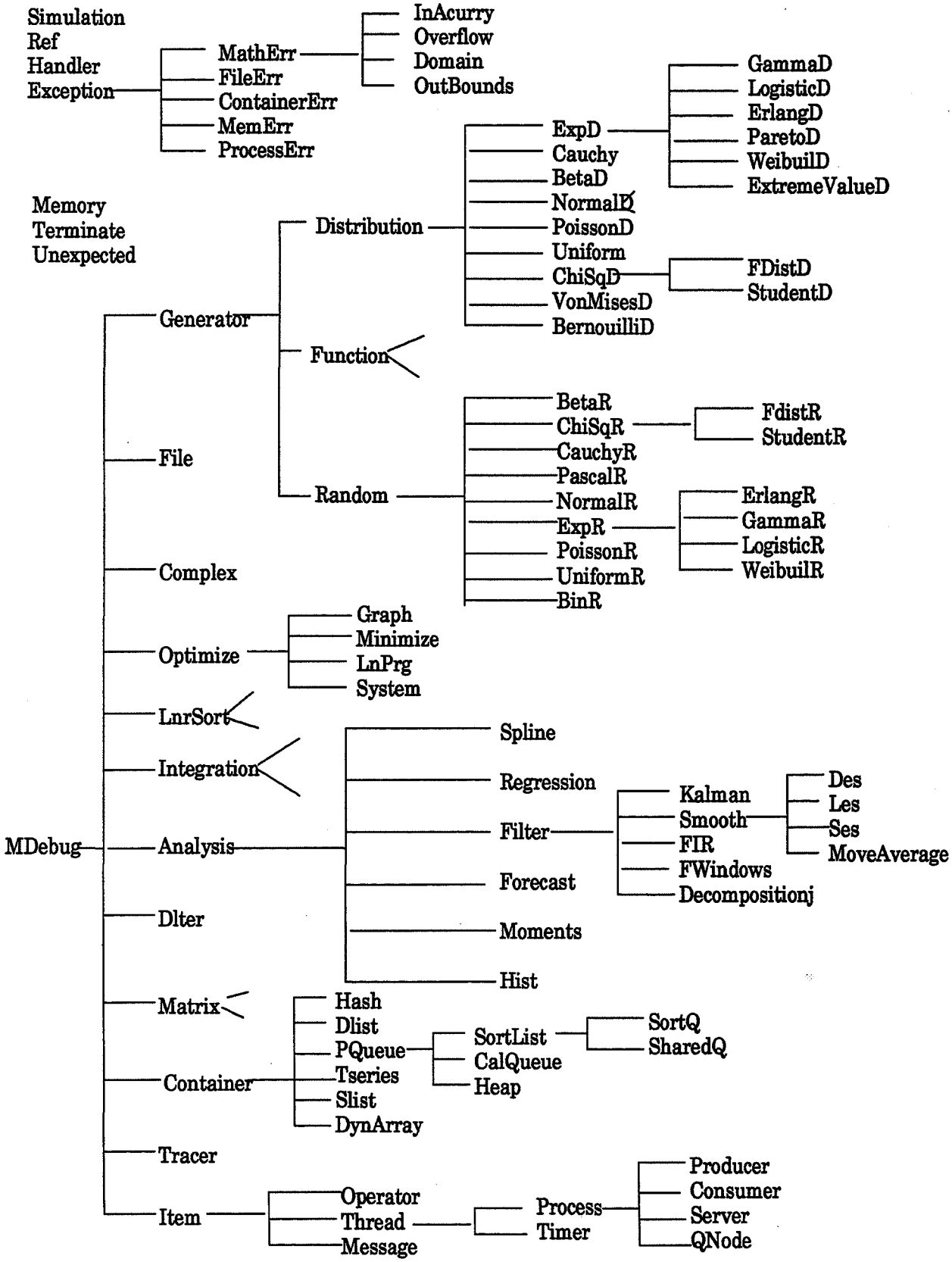


Figure 5-10 Hiérarchie des Classes d'Objets Prédéfinies par Meijin++

IV Simulation Orientée-Objets

IV.1 Environnement du Modèle de Simulation

Nous avons proposé, dans la section III du chapitre IV, une architecture de modèles de simulation constituée de trois modules: pré-simulation, simulation et post-simulation. Le module de pré-simulation est utilisé pour construire, configurer et documenter les modèles de simulation et les composants (classes d'objets) du modèle. Le module de simulation gère l'exécution du modèle et collectionne les résultats de sortie désirés qui seront analysés par les objets du module de post-simulation. Popken [POPKEN 92] a proposé une architecture de développement des modèles de la simulation (figure 5-11) qui nous semble bien adapter à notre structure de développement. Trois bases de données correspondantes doivent être associées avec ces modules: base de données de l'architecture du modèle qui constitue l'ensemble des classes d'objets du système, base de données du modèle de simulation courant qui constitue des objets expérimentaux de la simulation et base de données auxiliaires qui constitue des objets de statistiques et d'aides, etc., pour les modules pré/post-simulation.

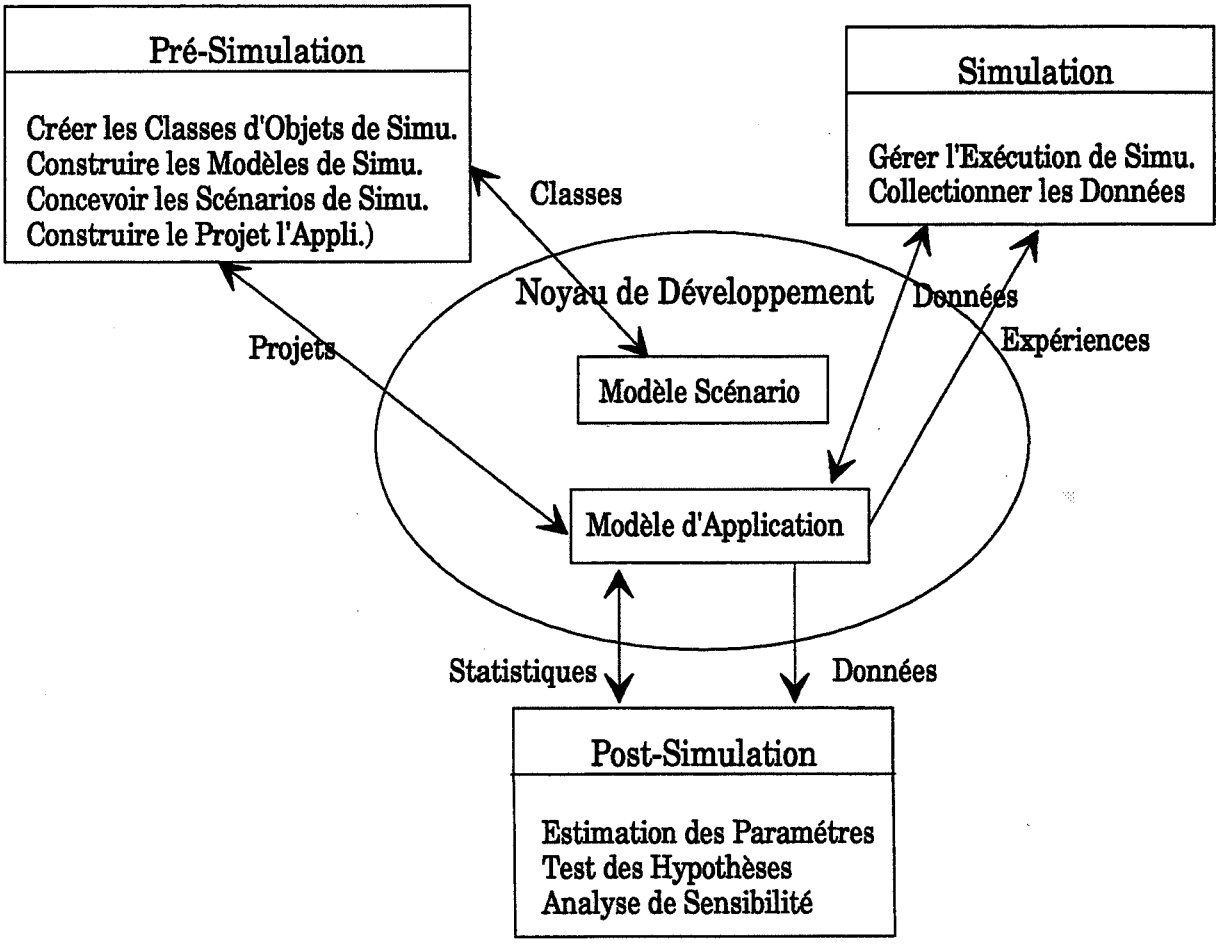


Figure 5-11 Architecture du Noyau de Développement de Modèles de la Simulation

Ce noyau de développement est décomposé en six étapes dans le progiciel Meijin++:

Etape 1: Initialisation de l'exécution en extrayant les données du scénario définies par le module de pré-simulation (choix de la structure de l'ordonnanceur, des primitives de commande, des générateurs de nombres aléatoires, etc.);

Etape 2: Instanciation des classes d'objets passifs (pièces, gammes, ateliers, etc.);

Etape 3: Instanciation des classes d'objets actifs (machines, transporteurs, processus de commande, stations, règles de management, etc.);

Etape 4: Exécution de la simulation;

Etape 5: Analyse des données et évaluation des performances;

Etape 6: Mise à jours de bases de données et destruction des objets du modèle de la simulation courante.

Une interface graphique avec des menus de contrôle facilite le travail des utilisateurs et réduit les lacunes entre les modèles conceptuels et ses implémentations. Puisque le modèle de simulation des systèmes de production est complexe, divers utilisateurs peuvent intervenir dans cette plateforme. Nous classifions ces intervenants en quatre niveaux d'utilisation (figure 5-12) [YE 93]: 1) niveau décideur, 2) niveau constructeur de modèles, 3) niveau développeur de classes du domaine et 4) niveau concepteur et programmeur de classes élémentaires. Au niveau décideur, peu d'expérience de la simulation est nécessaire, les décideurs ou les analystes modifient simplement des paramètres sur des modèles de simulation ou sur des outils statistiques prédéfinis. Au niveau 2, les constructeurs de modèles sélectionnent les classes d'objets disponibles contenues dans la librairie des classes et les instancient pour construire un modèle de simulation. Les modèles sont construits, testés et compilés pour être utilisés par les décideurs. Au niveau 3, les développeurs de classes définissent, testent et valident de nouvelles classes d'objets du domaine pour la simulation, ils enrichissent les classes d'objets sémantiques et les classes d'objets mathématiques et auxiliaires, et valident ces instances pour bien couvrir le domaine de l'application. Ces classes vont être utilisées au niveau 2. Enfin, au niveau 4, les programmeurs utilisent un langage de programmation pour construire des classes de base pour la simulation telles que les listes, les processus, le noyau de synchronisation des processus (scheduler), l'interface graphique, les objets statistiques, etc.

Les deux premiers niveaux correspondent aux tâches des utilisateurs qui modélisent et utilisent leur propre modèle de simulation en instanciant et unissant les classes d'objets disponibles. Les deux derniers correspondent aux tâches des développeurs ou concepteurs qui se concentrent sur la mise à jour des classes d'objets existantes et implémentent de nouvelles classes d'objets du domaine qui seront utilisées ultérieurement. Les classes d'objets du domaine (sémantiques) et de résolution du problème (niveau trois) ont été abordées dans les deux chapitres précédents, nous détaillons donc ici les liens entre les niveaux trois et quatre en nous basant sur le progiciel Meijin++ pour implémenter la hiérarchie des objets des systèmes de production.

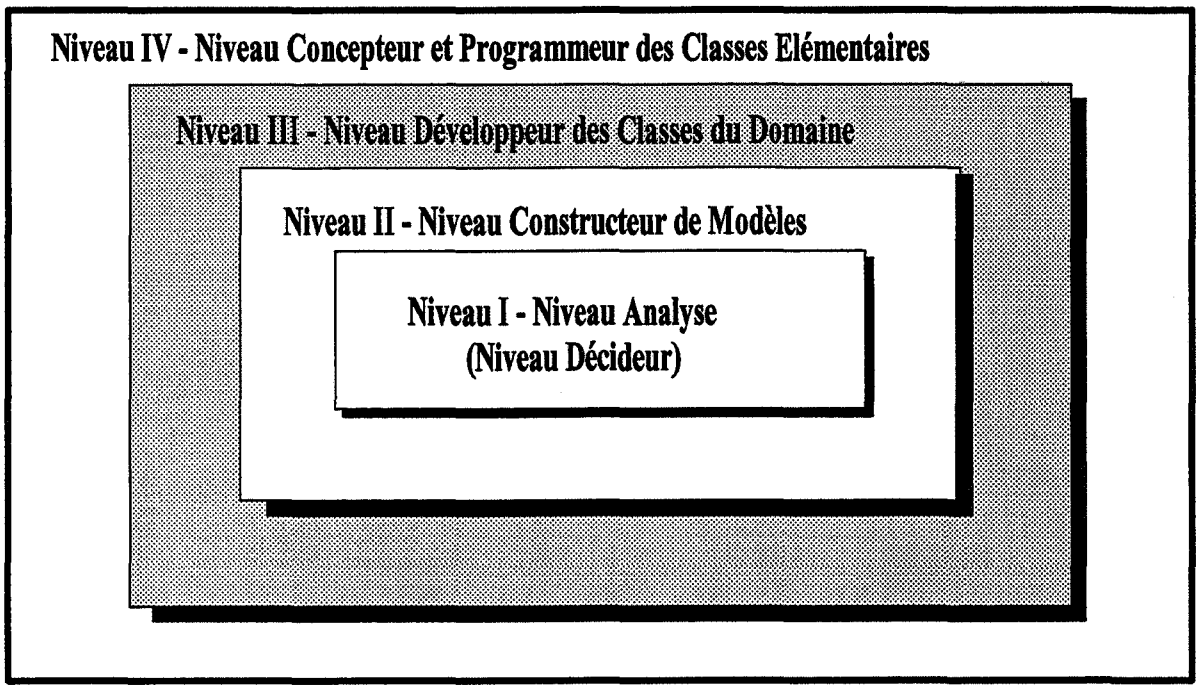


Figure 5-12 Niveaux d'Utilisation du Modèle de Simulation

IV.2 Hiérarchie des Classes d'Objets des Systèmes de Production

Avant de présenter la hiérarchie des classes d'objets de la production, quelques notions ou techniques de programmation pour la hiérarchisation des classes d'objets et son implémentation doivent être éclaircies. Il s'agit: de niveaux de modélisation pour déterminer les portés des attributs d'objets, des notions d'encapsulation, de persistance d'objets, de directives de préprocesseur, de constructeurs et de destructeurs des classes d'objets, etc.

Les constructeurs et les destructeurs déterminent la manière de créer, d'initialiser, de copier et de détruire les objets d'une classe. Un objet peut être créé statiquement (une déclaration de la classe) ou dynamiquement (utilisation de l'opérateur **new**), avec des arguments si on veut initialiser certains attributs ou sans arguments s'il s'agit d'un constructeur implicite. Nous

préconisons que chaque classe doive avoir un constructeur implicite qui fait appel à une fonction membre protégée *init*() pour initialiser les attributs structurels et conjoncturels d'objets en extrayant les valeurs d'une base de données ou à travers une interface d'interrogation graphique. Les directives du préprocesseur sont utilisées pour améliorer la performance de simulation du modèle. Par exemple, l'exécution d'un modèle avec la directive M_SUIVI est plus longue que celle sans suivi, mais elle fournit des indicateurs de performance de sortie plus complets et plus détaillés. Pour qu'on puisse faire l'exécution d'un modèle de la simulation de façon itérative, tous les objets sémantiques doivent être persistants, en d'autres termes, chaque classe d'objet est capable d'enregistrer (ou de lire) dans un fichier ces attributs concernés. La notion d'encapsulation sert à contrôler la visibilité des attributs d'objets, à modéliser les niveaux d'abstraction (droits d'accès des utilisateurs), etc.

IV.2.1 Hiérarchie des Classes d'Objets Actifs

Les classes d'objets actifs sont des classes dérivées directement ou par l' intermédiaire de l'objet *Process* du Meijin++. Une machine, par exemple, est héritée directement de l'objet Process avec des attributs décrits dans le modèle d'information de la machine (figure 3-22 du chapitre III). Les services identifiés lors de l'analyse sont concrétisés et implémentés par des méthodes correspondantes qui sont illustrées dans la figure 5-13.

delay(long) block(Thread*) pending_one(Thread**, unsigned) pre_empty() pending_all(Thread**, unsigned)	schedule() rank() exits(long) spawn() run()
select_fromQ() choisir_lot(Lot) choisir_mode_FAS(Mode) set_mode (Mode) recevoir() preparer () transformer () expedier(Pièce) traiter_panne() *méthodes pour manipuler* *les attributs* *etc*	add_temps_reparation(t) add_temps_usinage(t) add-temps-attente(t) add_temps_panne(t) add_compteur(i) valeur() ... ... temps-moyen-attente nombre-moyen-attente taux_occupation() taux_rebuts() loi_panne() nombre_lot_fabrique()

Figure 5-13 Méthodes de la Classe Machine

Les méthodes listées au-dessus de la ligne discontinue dans la figure 5-13 sont prédéfinies dans Meijin++. Les méthodes *delay, block, pre-empty, pending-one et pending-all* sont expliquées dans la section III.3. Notons que la méthode *pre-empty* ne peut pas être utilisée pour définir des processus pré-emptifs malgré la bonne volonté des concepteurs du Meijin++, puisque le mécanisme d'ordonnancement implémenté dans Meijin++ ne permet pas d'interrompre un processus thread (d'ailleurs, la notion thread n'autorise pas au fond d'installer des processus pré-emptifs). La méthode *schedule* sert à modéliser la sélection du prochain processus en état *éligible* et éventuellement l'avancement de l'horloge du système. La méthode *spawn* est utilisée pour activer un processus à l'état *non-opérationnel* et la méthode *exits* est une méthode surchargée de l'opérateur standard *::exits* du C++ pour des objets de type processus. Toutes ces méthodes systèmes concernant des primitives de création et de coopération des processus ne doivent pas être accédées et surchargées par les utilisateurs. Cependant, les méthodes *rank* et *run* sont des méthodes virtuelles qui seront en général redéfinir par les utilisateurs: la méthode *rank* sert à définir la priorité du processus dans l'objet ordonnanceur (schedule) en combinant le degré d'urgence du processus dans le monde réel et sa prochaine date d'événement; et la méthode *run* (que nous avons nommé *fonctionner* dans les objets dérivés de Process dans les chapitres précédents) est la méthode principale à surcharger par l'utilisateur pour définir le comportement temporel (cycle de vie) particulier de chaque machine.

Les méthodes listées au-dessous de la ligne discontinue dans la figure 5-13 sont des méthodes ajoutées au niveau de la classe machine:

- Les méthodes telles que add_temps_reparation, add_temps_usinage, add_temps_attente, etc., sont des méthodes accès du type *lire* et *écrire* (s'il y a des attributs correspondants);
- Les méthodes telles que temps_moyen_attente, taux_occupation, taux_rebuts, nombre_lot_fabrique, etc., sont des méthodes transformations du type calcul (sans génération d'événement asynchrone pour bloquer et débloquer les activités de la machine);
- Les méthodes telles que select_fromQ, choisir_lot, set_mode, etc., sont des méthodes virtuelles, appelées par des processus (méthodes bloquantes) de la machine, pour définir la gestion des activités (processus) en appelant des objets décisionnels concernés. Par exemple, la méthode select-fromQ modélise le choix du prochain lot (mais ne pas un article) à transporter d'une machine qui alimente la file d'attente amont de la machine courante à celle-ci. On peut, à l'intérieur de la méthode, appeler la méthode choisir-lot pour choisir des lots candidats à transporter s'il y a plusieurs lots prêts à être transportés, et la méthode select-fromQ va déterminer ultimement le lot à transporter selon la disponibilité des ressources et des transporteurs et lancer (activer) le processus du transporteur sélectionné.

- Les méthodes recevoir, preparer, transformer, expedier et tranter_panne sont des méthodes bloquantes ou asynchrones qui ont été expliquées précédemment dans les modèles de processus de la machine du chapitre IV.
- D'autres méthodes accès et test qui ne sont pas précisées dans la figure 5-13 comme la collection des données et les méthodes statistiques sont triviales. Elles seront définies selon les niveaux (détails) et l'objectif de la modélisation de la machine.

La classe *Station* est un objet dérivé ainsi de la classe Process du Meijin++ avec des méthodes en plus, telles que la gestion de ses serveurs, l'administration du temps de l'horloge, la gestion de panne des serveurs, la configuration des serveurs, etc. Les serveurs sont des machines qui effectuent réellement les opérations sur des articles; mais la gestion des processus des machines est encapsulée dans les processus de la station. En d'autres termes, la classe station est une abstraction (un processus logique) des n machines qui ont les mêmes comportements.

La classe d'objet *Transporteur* circule entre les n machines dans l'atelier. Elle a donc n files d'attente amont qui sont des files d'attente aval de machines de l'atelier et n files d'attente aval qui sont des files d'attente amont de machines (figure 5-14). Les méthodes ajoutées par rapport à la machine sont: les choix de trajet de transfert, la détermination de la taille de lot de transport, les choix de pièces à transporter s'il y a plusieurs pièces prêtes à être transportées, etc. Le tableau 5-1 liste les méthodes principales à installer dans l'objet Transporteur en termes du domaine. Cet objet est dérivé d'une classe d'objet *QNode* prédéfinie dans Meijin++.

mise en service	choisir le lot à transporter
acquérir des ressources	déterminer la taille du lot à transporter
acquérir un trajet du réseau	choisir le trajet de transfert
charger des articles	choisir des ressources
déplacer des articles	choisir la vitesse
déplacer à vide	
libérer des ressources	calculer le taux d'utilisation
traiter la panne	calculer le temps moyen d'attente
	calculer la distance du trajet
retourner la valeur actuelle du transporteur	calculer le temps de transfert
... ...	... ...
méthodes mise à jour des données	méthodes statistiques, etc.

Tableau 5-1 Méthodes d'un Objet Transporteur

Il faut préciser que l'utilisation de ces méthodes pour définir le comportement des transporteurs dans la méthode *fonctionner* est très subjective et différente selon le type de transporteurs: convoyeur, chariot à filo-guidé, chariot libre, etc. La figure 5-15 illustre le diagramme général des transitions d'état d'un objet transporteur. On voit ici que les méthodes impliquées sont limitées. Il n'y a pas des appels à des méthodes comme déterminer la taille du lot à transporter, choisir la vitesse, calculer la distance, etc. Cela dépend de l'application à modéliser.

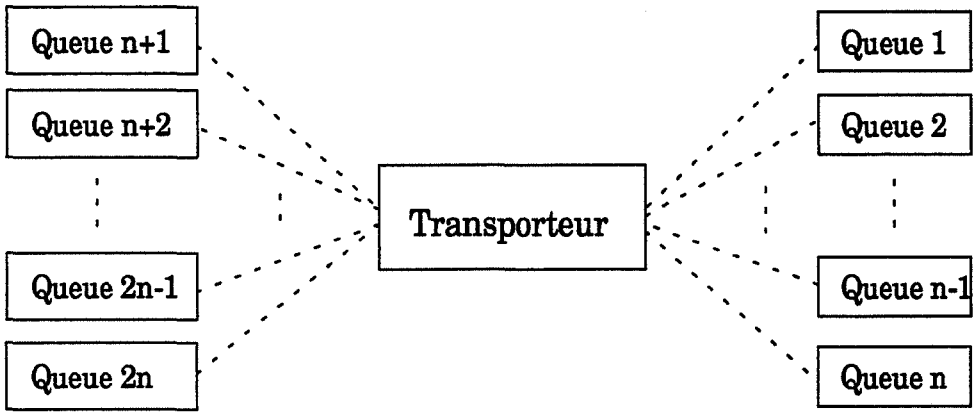


Figure 5-14 Structure de la Classe Transporteur

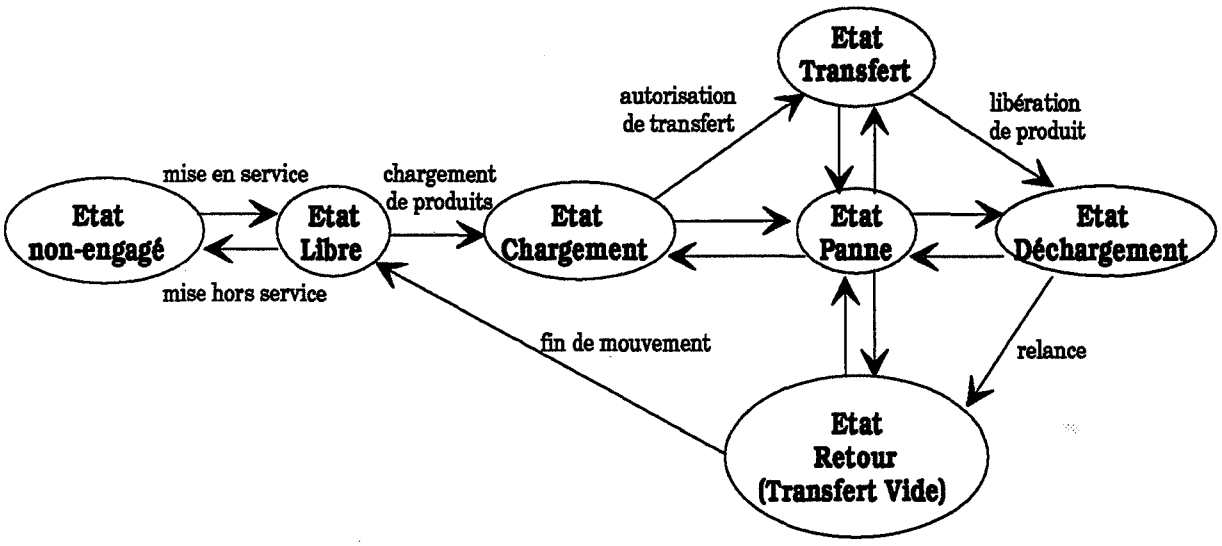


Figure 5-15 Diagramme des Transitions d'Etat d'un Transporteur

La gestion de panne, la gestion de temps de préparation et le contrôle technique d'un objet actif sont réalisés par des classes d'objets dérivées d'une classe *Timer* du Meijin++. Ces classes consistent à arrêter un processus pour un laps de temps. On obtient donc une hiérarchie des classes d'objets actifs présentée dans la figure 5-16 qui couvre les objets sémantiques minimum dans les systèmes de production. On voit qu'il n'y a pas l'objet atelier dans la hiérarchie des classes d'objets actifs. En effet, un objet atelier est un objet passif qui définit l'organisation des

machines, des stations et des transporteurs. Le comportement de l'atelier est manifesté par ses composants qui sont contrôlés par des règles de management définies pour cet atelier. Bien entendu, on peut définir une telle règle comme un processus logique qui conduit directement les processus physiques de chaque composant de l'atelier au lieu de les considérer comme des objets passifs (règles ou algorithmes). Mais comment peut-on les implémenter dans le sens informatique. Nous n'avons pas tenté de les implémenter avec le support du Meijin++.

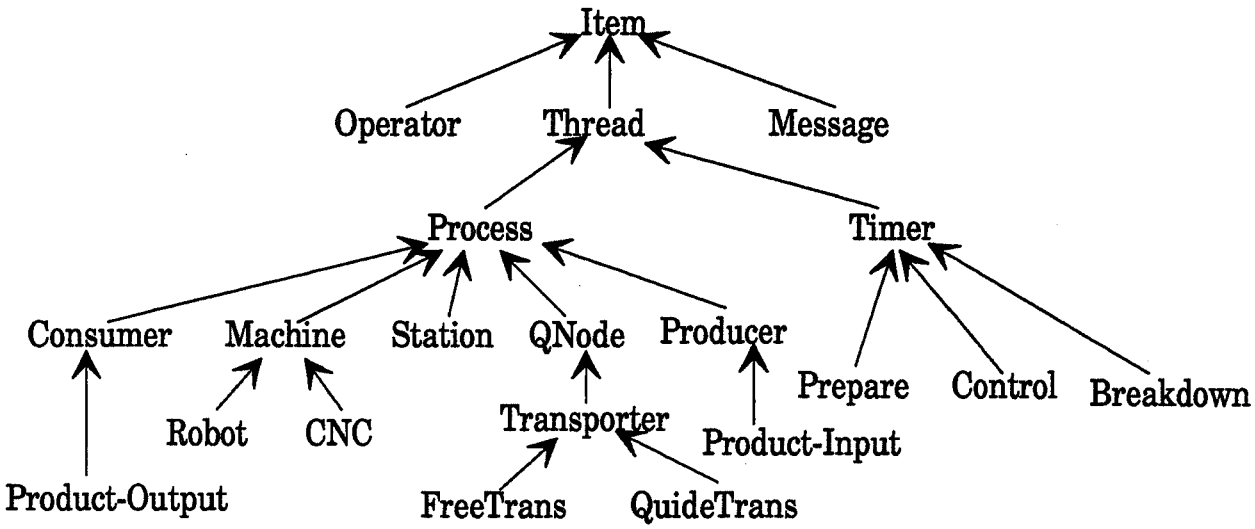


Figure 5-16 Hiérarchie des Classes d'Objet Actifs des Systèmes de Production

IV.2.2 Hiérarchie des Classes d'Objets Passifs

Nous avons présenté, pour chaque objet actif, les processus essentiels de la production. L'enchaînement de ces processus est conditionné (méthode *fonctionner*), d'une part par leur état, d'autre part par les flux de production (matière) et d'information. Les contraintes économiques et les critères de performance exigent moins la prise en compte de chacun de ces processus indépendants les uns des autres que leur bon enchaînement au sein de la notion de flux matière qui influence directement les performances de la production. En d'autres termes, il faut intégrer les classes d'objets de transaction (au niveau d'atelier du modèle) et des classes d'objets décisionnels (aux niveaux d'unité de production et de système de production du modèle) afin de construire un modèle global de la simulation. Ces deux types des classes d'objets sont des objets séquentiels ou passifs qui n'ont pas un comportement autonome.

Les classes d'objets passifs implémentées dans la librairie sont divisées en deux catégories: des classes d'objets dérivées de la classe d'objet *Container* et des classes d'objets dérivées de la classe *Item* qui seront stockées dans des classes conteneurs. La figure 5-17 illustre la hiérarchie des classes d'objets passifs implémentée avec le progiciel Meijin++. Les produits, les pièces et les objets manipulés par des objets actifs sont dérivés de l'objet de transaction *Item*. Les

différentes files d'attente qui arrangent les objets circulants sont dérivées de l'objet d'association *SortList*. La gamme (*OpList*) et la nomenclature (*PartList*) sont des objets persistants qui dérivent respectivement des objets de listes simples et doubles.

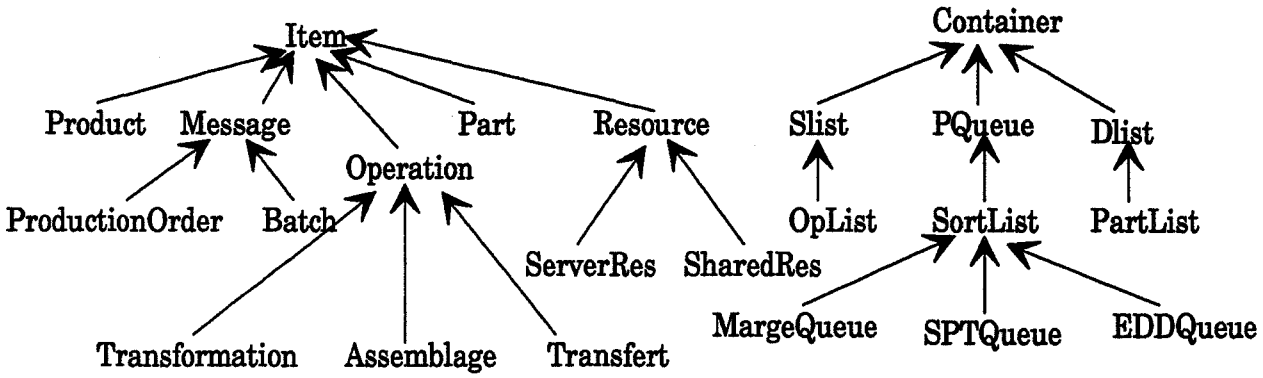


Figure 5-17 Hiérarchie des Classes Passives des Systèmes de Production

Nous détaillons ici seulement l'enchaînement des flux des attributs et des méthodes d'un objet pièce dans le modèle de simulation. La figure 5-18 liste le diagramme des transitions d'état d'une pièce dans l'atelier. La différence de ces états consiste dans le calcul des performances du système (coût de production, temps moyen d'attente, coût de stockage, etc.). Les transitions d'état de la pièce sont manipulées par des objets actifs tels que les machines et les transporteurs. Par exemple, quand une machine prend une pièce dans la file d'attente amont, la méthode *recevoir* de la machine va d'une part modifier l'état de la pièce: état *stock* ---> état *production*, d'autre part accéder aux attributs de la pièce comme l'accès du temps opératoire définis dans la méthode *tempsOpératoire* de la machine illustrée comme suit:

```
TIME Machine::TempsOpératoire () {
        Part* pièce;
        OpList gamme;
        Operation op;
        pièce = choisir_pièce(); //Sélectionner la pièce à fabriquer dans la FAE
        gamme = *(pièce->gamme); //accéder à la gamme de la pièce sélectionnée
        op = *(gamme(pièce->position));  //extraire l'opération courante de la pièce
        tempsOpératoire = op.tempsOpératoire; //accèder au temps opératoire
        return tempsOpératoire;
}
```

La relation entre la machine, la pièce, la gamme et l'opération est une implémentation de la relation de l'application. L'utilisateur n'oblige pas de connaître ces détails d'implémentation

(encapsulation). D'autres structures peuvent aussi être utilisées dans la modélisation. Cela dépend de l'outil choisi et de la préférence du concepteur des classes d'objets.

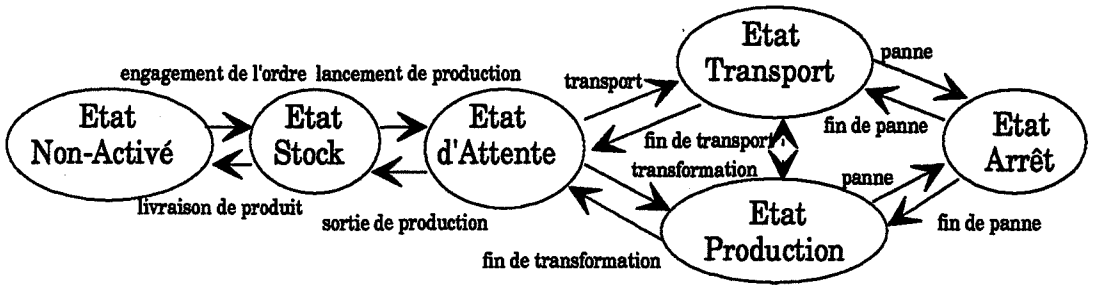


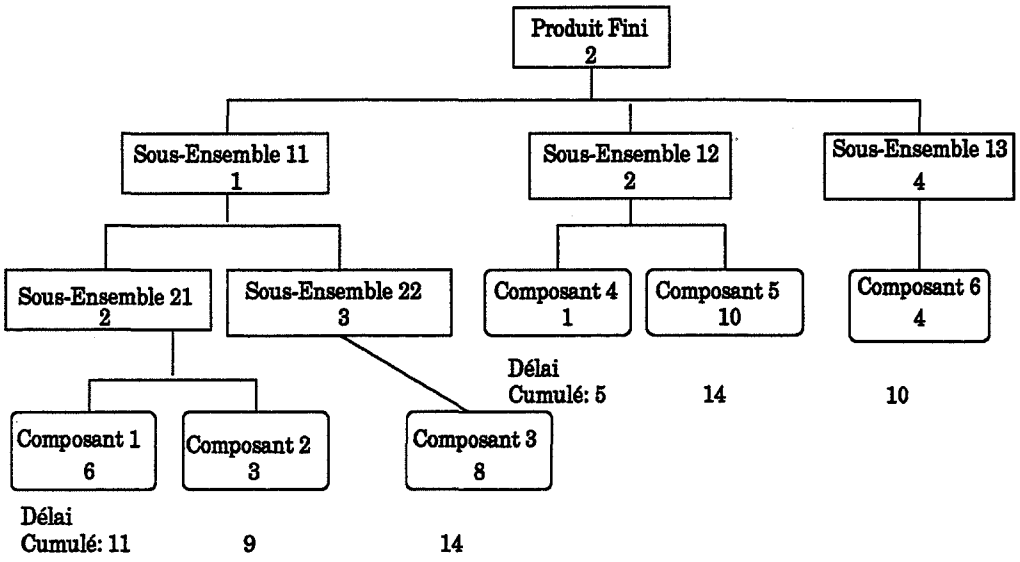
Figure 5-18 Modèle d'Etat d'une Pièce

Un autre exemple d'utilisation de l'objet comme la nomenclature qui est un objet association (arborescence) et qui est initialisée avant le lancement de la simulation par les attributs structurels et conjoncturels d'un objet produit (figure 5-19). Dans le module pré-simulation, on définit une procédure pour ordonner les dates de lancement de ces composants dans le temps; on obtient donc un schéma du type PERT. Les données déduites de ce calcul constituent les valeurs initiales des attributs conjoncturels des composants du produit pour définir les scénarios de simulation. Différents algorithmes peuvent être pré-installés dans la librairie. Lorsque l'utilisateur (niveau décideur) lance la simulation, il prend la responsabilité d'adopter les algorithmes qui lui conviennent. On dit aussi que c'est une pré-simulation dans ce cas.

IV.2.3 Utilisation des Classes d'Objets dans la Simulation Concurrente

Nous avons testé ces classes d'objets par différents modèles suivants (les programmes correspondants sont listés dans l'annexe 2):

- Modèle 1: une machine avec un générateur de pièces en amont et un consommateur en aval (pour produire un rapport de sortie, par exemple);
- Modèle 2: un variant de modèle 1 en ajoutant des opérations de retouche (un processus de feed-back pour la machine);
- Modèle 3: deux machines travaillent en parallèle avec un générateur de pièces commun et un consommateur en aval;
- Modèle 4: une réduction du modèle 3 dans lequel les deux machines sont remplacées par un objet station;
- Modèle 5: X machines ou stations en cascade (un atelier de type flow-shop);
- Modèle 6: X machines ou stations en interconnexion par les files d'attente (un atelier de type job-shop);
- Modèle 7: deux machines avec des ressources partagées, etc.



Nomenclature d'un Produit avec le Temps de Cycle Technique des Composants

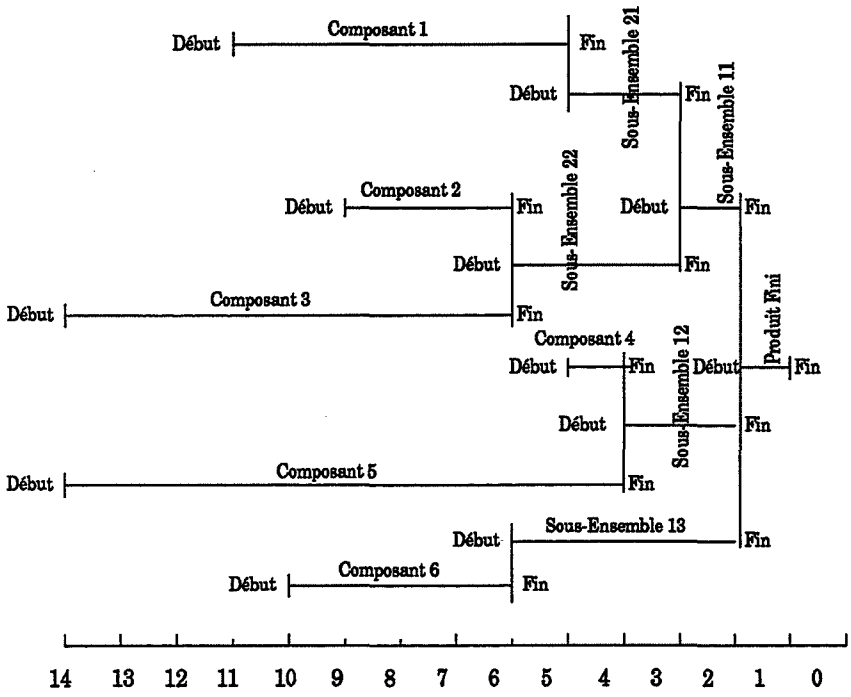


Schéma du PERT de la Nomenclature d'un Produit

Figure 5-19 Utilisation de l'Objet Nomenclature dans la Prise de Décision de Lancement

Nous avons surtout fait attention aux opérations de synchronisations temporelles entre les processus. Les résultats des modèles sont utilisés pour valider les méthodes synchrones installées dans les classes d'objets actifs. Deux types de décisions ont été testés avec ces modèles: les modèles 1, 2, 3 et 4 sont utilisés pour tester les règles de priorité (règles locales)

et les modèles 5 et 6 sont utilisés pour tester les règles locales et globales (règles de management). La gestion des ressources pour la synchronisation entre processus est illustrée par le modèle 7. L'objet actif transporteur est plus délicat à implémenter sans un modèle de simulation (une macro-structure du système) avec le support du Meijin++. Dès que la conceptualisation des processus de transporteur soit finie, nous essayons d'étendre notre modèle à la gestion de transporteur au niveau d'atelier.

V Conclusion

Longtemps l'informatique a traité des problèmes pour lesquels la réalité du temps qui s'écoule n'intervenait pas. Cette situation s'inverse maintenant parce que l'informatique entre dans la résolution de problèmes qui, par nature, ne peuvent ignorer cette réalité du temps (la simulation à événements discrets, par exemple). Dans ce chapitre, nous avons commencé par les concepts de base de la programmation concurrente: la notion de processus et les coopérations entre les processus en terme informatique. Ensuite, nous avons présenté trois langages ou outils basés sur le langage de programmation C++ qui permettent d'implémenter les modèles de simulation concurrente. Enfin, nous avons choisi un outil parmi ceux-ci pour illustrer comment nous avons implémenté les classes d'objets des systèmes de production.

Cependant, l'intervention du temps dans les modèles pose d'autres problèmes difficiles. Cela tient aux raisons suivantes [THORIN 90]:

1) **La difficulté de formalisation des problèmes temporels**. Cette difficulté vient, d'une part par la pauvreté d'expression de concepts temps réel dans des langages de programmation, d'autre part par le choix des modèles de la représentation des concepts temps réel (par exemple comment nous définissons des critères pour qualifier une entité du monde réel comme un objet processus [GOMMA 84] dans le modèle de simulation).

2) **Les conflits causés par des ressources limitées**. Cette difficulté est réelle et inévitable. On se rencontre ces conflits très fréquemment dans la vie courante. En effet, si le temps ne comptait pas, il serait concevable de traiter même un problème très complexe sur une machine quelconque; mais le temps est lui-même une ressource limitée, et l'impossibilité de sérier complètement l'emploi de la ressource temps conduit à des conflits d'emplois simultanés.

3) **La survenance aléatoire d'événements**. Le hasard est une notion subjective tenant à l'ignorance des causes ou du moins à l'impossibilité pratique d'en tirer des conclusions utiles. La difficulté est donc de se prémunir contre les conséquences de l'ignorance de certains événements aléatoires ou du moins de l'instant où ils surviennent.

Ce chapitre a eu pour présenter une synthèse des problèmes rencontrés dans la simulation des systèmes de production et de leurs solutions informatiques, à savoir:

* la formalisation des objets actifs (comportements temporels des processus);
* la communication et la synchronisation entre objets actifs (des conflits entre objets);
* et la prise en compte de l'aléatoire (dynamicité des processus).

Cependant la plupart des langages de simulation ne fournissent pas de moyens pour exprimer naturellement ces concepts (ils ont détourné la représentation de problèmes par certains éléments ou précisément certains blocs comme on les trouve dans les langages SIMAN, SLAM, GPSS, etc.). Les langages à objets sont choisis par leurs nombreuses qualités dans le génie logiciel, mais les concepts temps réel n'ont pas encore été intégrés dans la plupart des langages à objets. Les objets sont accessibles via des méthodes définies par une classe ou un prototype. Dans le modèle à objets concurrents, chaque objet possède son propre flot de contrôle. L'extension des langages à objets en des langages à objets concurrents (communicants) est proposée dans la littérature ([CAROMEL 93], [MEYER 93], [THOMAS 92], [ISHIKAWA et al. 92]) pour les problèmes d'implémentation. Nous avons présenté, parmi ceux issus du langage C++ que nous connaissons, ceux qui répondent à la simulation à événements discrets par l'approche processus: MODSIM II, SIM Plus Plus et Meijin++. Enfin nous avons choisi Meijin++ dans lequel chaque processus est un objet actif qui possède des primitives de création de processus et des mécanismes de communication et de synchronisation entre eux comme un outil de base pour construire la hiérarchie des classes sémantiques pour la simulation des systèmes de production.

Pour conclure, l'utilisation des langages à objets concurrents pour la programmation des modèles de simulation des systèmes de production est puissante et attractive. Deux approches existent pour unifier les concepts objets et concurrence dans les langages de simulation à objets:

* dans la première approche, où la notion de processus est orthogonale à celle d'objet, un processus doit s'attendre à chaque instant de son existence à être interrompu par un autre processus comme nous avons présenté dans les langages MODSIM et SIM Plus Plus.

* dans la seconde approche, où les processus et les objets sont synonymes, les processus interagissent en des points parfaitement déterminés: là où un message est émis et là où il est reçu comme on le trouve dans l'outil Meijin++.

Cependant, la combinaison de la concurrence et de l'héritage est complexe, car les méthodes des classes dérivées peuvent invalider les protocoles d'accès décrits dans les méthodes des classes supérieures. De nombreux auteurs concluent à l'impossibilité de concilier concurrence et héritage [AMERICA 89]. D'autres autorisent l'héritage en restreignant les méthodes héritées et/ou la localisation des instructions de synchronisation. Dans Meijin++, c'est la méthode virtuelle *run (fonctionner)* qui sert à définir les activités séquentielles pour les objets actifs, et elle est la seule à pouvoir être redéfinie pour les coopérations temporelles avec d'autres objets actifs dans une classe dérivée.

Il faut noter que une tendance de plus en plus forte à une conception de langage multi-paradigmes: procédure, déclarative, logique, parallèle/distribué, objet, etc., va certainement modifier et faciliter la simulation à objets concurrents. Nous ne avons pas présenté ces aspects dans la mémoire, car il dépasse notre cadre de travail.

Conclusions et Perspectives

I Résumé du Travail

Depuis l'avènement de Simula et de Smalltalk dans le développement de logiciels dans divers domaines, de nombreuses recherches (méthodes d'analyse et de conception, langages de programmation) ont été effectuées afin de rendre les applications à objets plus évoluées et plus fiables et les développements spécifiques moins coûteux et plus évolutifs.

Tout chercheur dans ce domaine sait que les systèmes à objets ne naissent pas dans un monde vide. C'est pourquoi les méthodes, les langages et les outils évoluent pour réutiliser et rendre réutilisables les développements entrepris, sous la forme de bibliothèques de classes. Une pression immense sur l'élaboration de normes d'objets conduit la plupart des fournisseurs de classes vers la standardisation d'objets [SNYDER 93]. Dans l'avenir, nous pouvons imaginer que nous trouverons sur le marché les briques de composants logiciels qui nous conviennent. Mais les éléments spécifiques d'un domaine ou d'un métier seront toujours développés sur mesure. C'est dans cette optique que nous avons effectué notre recherche: essayer d'appliquer les technologies objets pour la modélisation et surtout pour la simulation des systèmes de production.

Après avoir survolé les systèmes de production et les méthodes de la gestion de production dans le chapitre I, nous constatons que les systèmes de production sont des systèmes complexes, dont la compréhension et la gestion (prévision) de comportement passent nécessairement par une étape de modélisation. Car c'est à partir d'une observation de l'existant, dont on analyse la complexité en identifiant les composants représentatifs des systèmes manufacturiers qu'on élabore les modèles de ces systèmes.

Modéliser la structure des systèmes permet de mieux comprendre ce qu'ils sont. Des modèles de structure (SADT, MERISE, GRAI, CIMOSA) ont été présentés dans le chapitre II. Or pour un système complexe tels que les systèmes manufacturiers, il ne suffit plus de faire une analyse du fonctionnent (méthode SADT et GRAI) de chacun de ses composants avant de les assembler (méthodes MERISE et CIMOSA), il faut modéliser le comportement global et local des composants avant d'affecter un rôle à chacun de ces composants pour une application

particulière, c'est-à-dire qu'il faut décrire comment les composants (objets) doivent fonctionner avant d'expliquer comment un système (modèle) fonctionne. Cela nous conduit à construire des modèles de simulation des systèmes de production en utilisant l'approche par objet.

Les méthodes d'analyse et de conception à objets se sont multipliées ces dernières années. Cette profusion s'explique par le fait qu'aucune approche n'est vraiment satisfaisante ni ne permet de couvrir l'ensemble des besoins. En s'inspirant des techniques de l'analyse du domaine et de la méthode SHLAER/MELLOR, nous divisons l'étape d'analyse en deux sous-étapes: d'abord l'analyse du domaine dans laquelle on établit le contexte du développement (service, application, architecture et implémentation) et les classes d'objets (physiques, rôles, incidents, interactions et spécifications) du domaine; ensuite l'analyse de l'application qui se concentre sur une application particulière afin de transformer les modèles du domaine (communication, transition d'état, information) en des modèles de l'application. Ce qui constitue le chapitre III.

Dans le chapitre IV, nous sommes passés par une étape de conception de haut-niveau dans laquelle nous avons construit une architecture de modèles de simulation intégrés (composés de trois modules: pré-simulation, simulation et post-simulation), ouverts (divisés en quatre niveaux de modélisation: ressource, atelier, unité de production, G.P.A.O.) et flexibles (pour l'intégration des objets décisionnels), puis par une étape de conception de bas niveau: conception des classes. Nous avons conceptualisé une architecture ouverte et flexible d'objets actifs et passifs. Les structures et méthodes de classes d'objets sémantiques essentielles pour les systèmes de production (transactions, moyens de production et décisions) ainsi que celles des classes d'objets de résolution (utilitaires/auxiliaires, mathématiques, interface, application, etc.) sont présentées et conceptualisées. Enfin, le modèle d'objet actif est illustré à travers le comportement de deux classes d'objets sémantiques de systèmes de production: machine et station.

Le chapitre V est consacré à l'implémentation de modèles de simulation concurrente à objets. En commençant par un rappel des concepts de la programmation concurrente: la notion de processus et les coopérations entre les processus en terme informatique, nous avons étudié trois langages ou outils basés sur la notion de processus léger: MODSIM II, SIM Plus Plus et Meijin++, qui permettent d'implémenter les modèles de simulation concurrente. Enfin, les classes d'objets sémantiques des systèmes de production concernant surtout la partie physique sont hiérarchisées et implémentées à l'aide d'un outil de simulation Meijin++.

II Contribution

Techniquement, l'approche à objets est viable et concurrence dans toutes les étapes du cycle de vie des logiciels, depuis l'analyse et la conception jusqu'à la maintenance, les méthodes dites classiques. Tous les utilisateurs actuels de l'approche à objets s'accordent à vanter la facilité de modélisation qu'elle apporte et le temps gagné sur les charges de développement et de maintenance applicative: granularité fine, réutilisabilité maximale, encapsulation du comportement, unité de communication. Tout serait faisable avec de la programmation classique, mais avec un grand niveau de complexité.

Réutiliser plutôt que refaire. Mais une librairie des classes d'objets représente une structure hiérarchique d'un domaine et non pas une boîte à outils remplie de choses disparates. Bien que des classes d'objets standards existent (d'ailleurs c'est un critère très important pour instaurer un niveau de confiance suffisant dans l'industrie) dans la plupart des domaines [GAL 94], l'offre d'une librairie des classes d'objets qui soit suffisamment générale et complète pour la simulation des systèmes manufacturiers n'est, à notre connaissance, pas encore disponible (ou plutôt non assez crédible). Une bonne librairie des classes doit fournir un guide et une direction pour la conception et le développement du modèle de simulation d'une application. C'est pour cela que nous avons introduit une étape de l'analyse du domaine dans le développement à objets. Le but de l'analyse est d'essayer d'aboutir à un consensus sur le vocabulaire et la théorie du domaine de la production. Un modèle de simulation est constitué de quatre sous-domaines: application, service, architecture et implémentation, dans lesquels cinq types d'objets interagissent: physiques, rôles, incidents, interactions et spécification. Dès qu'un utilisateur désire utiliser une des classes dans la librairie, il doit avoir accès à l'ensemble des classes de la librairie à travers quatre modèles d'objets: modèle de communication, modèle d'état, modèle d'information et modèle de processus. Chacun des objets fonctionnels de la librairie s'intègre dans une perspective homogène et les composants interagissent de nombreuses manières.

La plupart des méthodes de conception à objets permettent de définir des objets séquentiels, c'est-à-dire, des objets qui fournissent une encapsulation d'organisation (structure) et certains services prêts à être utilisés. Or la simulation concurrente est un modèle dynamique: différents objets peuvent exécuter diverses activités en même temps ou un objet peut exécuter plusieurs activités concurrentiellement. Nous avons introduit les concepts agents dans la conception des classes d'objets. Le comportement des agents est défini dans une méthode particulière *fonctionner* par ses quatre types d'activités (méthodes): perception, cognition, décision et action. L'objectif de considérer un objet comme un agent est de décrire comment l'objet doit se comporter (e.x., évolution des attributs d'objets: structurels, conjoncturels, instantanés, événementiels, historiques et statistiques), mais non pas d'expliquer comment il fonctionne (entrées/sorties) dans le système à objets. Nous pouvons construire des modèles à objets

distribués et évolués avec ces agents (objets autonomes) qui répondent bien à la problématique posée dans l'introduction (modifiabilité des modèles de systèmes de production et flexibilité de gestion de production).

La troisième contribution concerne l'implémentation de modèles de la simulation concurrente. L'intégration de la concurrence et de l'héritage semble être une tâche très difficile. Le code des synchronisations exprimé à un niveau de la hiérarchie de classes ne peut prendre en compte les méthodes ajoutées et les redéfinitions qui apparaissent à un niveau inférieur de cette hiérarchie. Afin de résoudre ce conflit, les propositions existantes restreignent les objets actifs à n'avoir qu'une activité au plus en leur flot de contrôle interne. La notion de thread est utilisée pour implémenter la séquentialité de l'activité des objets concurrents qui s'exécutent en quasi-parallèle dans un modèle de communication inter-processus à mémoire commune. Une hiérarchie des classes d'objets actifs et passifs pour la simulation concurrente des systèmes de production est proposée et installée (ou en cours d'installation).

III Perspectives

Les propositions faites dans ce mémoire constituent un premier pas vers la simulation concurrente à objets pour les systèmes manufacturiers. Un système de production est composé de trois sous-systèmes: un sous-système opérant, un sous-système d'information et un sous-système décisionnel. Nous avons construit des classes d'objets essentielles pour le sous-système opérant de la production en insistant sur la notion de processus avec l'esprit d'ouverture pour intégrer des décisions dans les modèles de simulation. Mais les décisions modélisées dans ce mémoire sont des décisions locales représentées essentiellement par des règles de priorité; une extension des structures des classes décisionnelles comprenant des décisions globales (règles de management, par exemple) flexibles et ouvertes est tout à fait envisageable. Réellement, pour les technologies objets comme pour toute nouvelle technique, c'est le premier pas qui coûte le plus en investissement. Les concepts tels que les méthodes virtuelles, les classes abstraites, les types génériques ou les modèles, la résolution tardive sont suffisant pour représenter et modéliser les décisions complexes dans la production; mais il faut avoir une vision globale et pertinente de la gestion de production avant de les structurer en des classes d'objets et hiérarchiser ces classes dans une librairie comme nous l'avons fait avec les classes d'objets physiques des systèmes de production. Faute de connaissance sur la gestion de production et de temps de développement, nous n'avons pas pu nous investir plus dans cet aspect. Une thèse en cours dans l'équipe traite l'aspect de la structuration des classes d'objets de gestion et de diagnostic de la production dans les modèles de simulation concurrente à objet [OUZ 94].

La *complexité* consiste, dans l'approche à objets, d'une part dans l'identification et la spécification d'objets, d'autre part dans l'unification des objets. Complexité est la somme de *diversité* (multiplicité des objets) et *complication* (relations trop nombreuses entre les objets pour être appréhendées en même temps par l'esprit humain). La démarche de conception d'un objet en différents niveaux: objet passif, objet actif (concurrent), objet autonome (agent, contrôle), objet décideur, nous semble une voie prometteuse pour réduire la complexité du système à objet. A cause de la difficulté d'implémentation au sens informatique, les problèmes de l'intégration des objets décideurs dans les modèles de simulation ne sont pas bien résolus (ou on ne les a pas abordés) dans ce travail. Les objets ont une dimension d'ordre sociologique. Bien que l'objet est une façon de penser avant d'être une technique, la question n'est pas de savoir si nous allons vers l'objet, mais plutôt comment appliquer ces techniques dans la modélisation et la simulation des systèmes de production tant au niveau de la conception qu'au niveau de l'implémentation. Une intégration des concepts multi-agents issus de l'intelligence artificielle distribuée et des systèmes multi-agents peut résoudre des problèmes d'intégration des objets décideurs dans les modèles de la simulation concurrente à objets des systèmes de production.

Enfin, une amélioration des hiérarchies des classes d'objets sémantiques (attributs et méthodes) pour la performance et la compréhension, et l'enrichissement des classes d'objets décisionnels pour que la librairie soit plus complète, s'inscrivant dans le cadre d'une démarche de modélisation des systèmes industriels de l'équipe "Etude et Modélisation des Systèmes Industriels" du Centre SIMADE pour aboutir à un système d'aide à la décision ouvert et flexible, constituent les principales activités courantes autour d'un produit en développement appelé SIM'3.

REFERENCES

[ADAMA 84] ADAMA, F.B. "*Contribution à un Analyseur Informatique GRAI*". Thèse Doct.: Université de Bordeaux I, 1984. 149p.

[AKERBAEK 93] AKERBAEK, T. "C++, Coroutine, and Simulation". *The C User Journal*, 1993, No. 3, p. 74-86.

[AMAR et al. 90] AMAR, S., CASTELAIN, E. et GENTINA, J.C. "Modélisation des Moyens de Production par Langages Orientés Objet en vue de la Conception de la Commande d'un Système de Production Flexible". *Colloque International sur Productique & Intégrations*, Bordeaux, juin 12-14 1990. p.325-334.

[AMERICA 89] AMERICA P. "Issues in the Design of a Parallel Object-Oriented Language". *Formal Aspect of Computing*, 1989, Vol. 1, No.3, p. 366-411.

[AMICE 89] AMICE CIMOSA "*Open System Architecture for CIM*". Berlin: Springer_Verlag, 1989. 212p.

[ANDREWS et al. 83] ANDREWS, G.R. et SCHNEIDER, F.B. "Concepts and Notations for Concurennt Programming". *Computing Surveys*, 1983, Vol. 15, No. 1, p. 3-43.

[AYEL 91] AYEL, J. "CIMES, Un Système d'Intelligence Artificielle Distribuée pour la Supervision en Continu des Activités de Gestion de Production". Thèse Doct.: Université de Savoie, 1991. 228p.

[BAILLY et al. 87] BAILLY, C., CHALLINE, J.F., FERRI, H.C., GLOESS, P.Y. et MARCHESIN, B. "*Les Langages Orientés-Objets*". Toulouse: CEPADUES-Editions, 1987. 223p.

[BALCI 90] BALCI, O., "Guidings for Successful Simulation Studies". *Proceeding of the 1990 Simulation Conference*, New Orleans, Louisiana, decembre 9-12, 1990. p.25-32.

[BARNES 88] BARNES, J. "*Programmer en ADA*". Paris: InterEditions, 1988. 495p.

[BARAKAT 91] BARAKAT, O. "*Contribution à la Modélisation et à la Simulation Orientées-Objets des Systèmes Flexibles de Production*". Thèse Doct.: Université de Franche-Compté, 1991. 156p.

[BARBIER] BARBIER, F. "*Une Approche Objet Dédiée à la Fonction Production d'une Entreprise Manufacturière: Application à la Gestion de Production*". Thèse Doct.: Université de Savoie, 1989. 208p.

[BEL et al. 85] BEL, G. et DUBOIS, D. "Modélisation et Simulation de Systèmes Automatisés de Production". *Revue APII*, 1985, vol. 19, No. 3, p.3-43.

[BEL et al. 88] BEL, G., BENSANA, E. et DUBOIS, D. "Construction d'Ordonnancements Prévisionnels: un Compromis entre Approches Classiques et Systèmes Experts". *2ème Conférence Internationale Systèmes de Production*, Paris, avril 6-10, 1988. p.709-726.

[BEL 88] BEL, G. "Ordonnancement et Intelligence Artificielle". *Colloque International Productique et Robotique: les Apports de l'Intelligence Artificielle*, Bordeaux, mars 15-17, 1988. p.4-11.

[BEL et al. 90] BEL, G. et CAVAILLE, J.B. "Intégration de la Simulation dans la Conception de Systèmes de Production: Avantages et Dangers de l'Approche par Langages à Objet". *Colloque International sur Productique & Intégrations*, Bordeaux, juin 12-14 1990. p.597-603.

[BEN-ARI 86] BEN-ARI, M. "*Processus Concurrents: Introduction à la Programmation Parallèle*". Paris: Masson, 1986. 174p.

[BELANGER 90a] BELANGER, R. "*MODSIM II Reference Manual*". CACI Products Company, La Jolla, CA, 1990. 336p.

[BELANGER 90b] BELANGER, R. "MODSIM II - A Modular, Object-Oriented Language". *1990 Winter Simulation Conference Proceeding*, New Orleans, USA, decembre 9-12. 1990. p.118-122.

[BENASSY 90] BENASSY, J. "*La Gestion de Production*". Paris: Hermes, 1990, 251p.

[BENSLEY et al. 92] BENSLEY, E.H., GIDDING, U.T., LEIVENT, J.I. et WATRO, R.J. "A Performance-Based Comparason of Object-Oriented Simulation Tools." The MITRE Cooperation, Bedford, MA, 1992. 86p. Rapport Intern.

[BERANGER 87] BERANGER, P. "*Les Nouvelles Règles de la Production. Vers l'Excellence Industrielle*". Paris: Dunod Entreprise, 1987. 212p.

[BERSHAD et al. 88] BERSHAD, B.N., LAZOWSKA, E.D. et LEVY, H.M. "PRESTO: a System for Object-Oriented Parallel Programming". *Software-Pratice and Experience*, 1988, Vol. 18, No. 8, p.713-732.

[BELT et al. 86] BELT, B. et BRUN, F. "*La Méthode MRP-II: Gérer l'Interface Commercial et Production*". Paris: Cabinet BILL BELT S.A., 1986.194p.

[BINDING 85] BINDING, C. "Cheap Concurrency in C". *SIGPLAN Notices*, 1985, Vol. 20, No. 9, p.21-26.

[BIRTWISTLE et al. 73] BIRTWISTLE, G., DAHLO, O., MYHRHAUG, B. et NYGAARD, K. "*SIMULA Begin*". New-York: Petrocelli Charter, 1973. 125p.

[BLANCHARD 79] BLANCHARD, M. "*Comprendre, Maîtriser et Appliquer le GRAFCET*". Toulouse: Cepadus édition, 1979. 194p.

[BORGEN et al. 89] BORGEN, E. et STANDHAGAN, J. "An Object-Oriented Tool Based on Discrete Event Simulation for Analysis and Design of Manufacturing Systems". *Optimization of Manufacturing Systems Design*, Edité par D.L. SHUNK, Amsterdam: North-Holland, 1989. p.195-220.

[BORLAND 93] "*ObjectWindows pour C++: Guide de l'Utilisateur*". Velizy Villacoublay: Borland Internatonal, Inc., 1993. 449p.

[BOOCH 92] BOOCH, G. "*Conception Orientée-Objets et Applications*". France: Addison-Wesley, 1992. 588p.

[BOOCH 93] BOOCH, G. "Développement Logiciel Orienté Objets: La Méthode Bouch Processus et Pragmatique". *Génie Logiciel & Systèmes Experts*, 1993, No. 30, p.4-13.

[BOUKACHOUR et al. 93] BOUKACHOUR, J., GALINHO, T. et PECUCHET, J.P. "Etude Comparative entre Simulation et Placement en Ordonnancement". *Revue APII*, 1993, Vol. 27, No. 5, p.561-585.

[BOURDICHON 93] BOURDICHON, P. "La Modélisation en Entreprise CIM-OSA et Ingénierie Simultanée: L'Ingénierie Simultanée dans le Cycle de Vie des Projets". *Colloque Université d'Eté du Pôle Productique Rhône-Alpes*, sept. 13-17, 1993. Aussois, France, 38p.

[BRAMS 83] BRAMS, G.W. "*Réseaux de Pétri: Théorie et Pratiques*". Paris: Masson, 1983. Tome 1, 184p.

[BRAESCH 89] BRAESCH, C. "*Approche de Modélisation du Système de Production d'une Entreprise Manufacturière*". Thèse Doct.: Université de Franche-Comté, 1989. 202p.

[BRODIER 88] BRODIER P.L. "Une Autre Approche de la Gestion: la Valeur Ajoutée Directe". Paris: AFNOR Gestion", 1988. 165p.

[BROWN 78] BROWN, M.R. "Implementation and Analysis of Binomial Queue Algorithms". *SIAM J. Comput.*, 1978, Vol. 7, No. 3, p.298-319.

[BROWN 88] BROWN, R. "Calendar Queues: A Fast 0(1) Priority Queue Implementation for the Simulation Event Set Problem". *Commun. of ACM*, 1988, Vol. 31 No. 10, p.1220-1227.

[BUHR et al. 90] BUHR, P.A. et STROOBOSSCHER, R.A. "The μSystem: Providing Light-Weight Concurrency on Shared-Memory Multiprocessor Computers Running UNIX". *Software-Pratice and Experience*, 1990, Vol. 20, No. 9, p.929-964.

[BUHR et al. 92] BUHR, P.A., DITCHFIELD, G., STROOBOSSCHER, R.A. et YOUNGER, B.M. "μC++: Concurrency in the Object-Oriented Language C++". *Software-Pratice and Experience*, 1992, Vol. 22, No. 2, p.137-172.

[CANALS 86] CANALS, D. "*Ordonnancement d'Atelier par Simulation: Etude des Règles de Priorité et Aide au Lancement*". Thèse Doct. Ing.: Ecole Nationale Supérieure de l'Aéronautique et de l'Espace de Toulouse, 1986. 326p.

[CARLIER et al. 88] CARLIER, J. et CHRETIENNE, P. "*Les Problèmes d'Ordonnancements: Modélisation, Complexité, Algorithmes*". Paris: Masson, 1988. 326p.

[CAROMEL 93] CAROMEL, D. "Toward a Method of Object-Oriented Concurrent Programming". *Commun. of the ACM*, 1993, Vol. 36, No. 9, p.90-102.

[CASTELLANI 93] CASTELLANI, X. "Méthodologie Générale d'Analyse et de Conception des Systèmes d'Objets: l'Ingénierie des Besoins". Paris: Masson, 1993. 402p.

[CAVAILLE et al. 87] CAVAILLE, J.B. et PROTH, J.M. " *SIPRODIS: Pratique de la Simulation en Production Discontinue*". Nanterre: Colloques & Conseil, 1987. 333p.

[CHILOUP-BEKAEPT 91] CHILOUP-BEKAEPT, M.H. "*Utilisation de la Notion d'Objets avec Contraintes pour la Modélisation et la Simulation des Systèmes de Production*". Thèse de l'Université de Lille Flandres Artois, 1991. 154p.

[COAD et al. 90] COAD, P. et YOURDON, E. "*Object-Oriented Analysis*". Englewood Cliffs, N.J.:Prentice-Hall, 1990. 232p.

[COAD et al. 91] COAD, P. et YOURDON, E. "*Conception Orientée-Objets*". Englewood Cliffs, N.J.:Prentice-Hall, 1991. 195p.

[COMMI 90] COMMISSARIAT GENERAL DU PLAN "*L'Usine du Futur: l'Entreprise Communicante et Intégrée*". Paris: la documentation française, 1990. 218p.

[COMMUNI 90] COMMUNI. OF ACM "Object-Oriented Design and Programming". *Communi. of ACM (Numéro Spécial)*, 1990, Vol. 33, No. 9, p.35-146.

[COMMUNI 92] COMMUNI. OF ACM "Analysis and Modeling in Software Development". *Communi. of ACM (Numéro Spécial)*, 1992, Vol. 35, No. 9, p.35-172.

[COMMUNI 93] COMMUNI. OF ACM "Concurrent Object-Oriented Programming". *Communi. of ACM (Numéro Spécial)*, 1993, Vol. 36, No. 9, p.34-137.

[CRIBBS et al. 92] CRIBBS, J., MOON, S. et ROE C. "*An Evaluation of Object-Oriented Analysis and Design Methodologies*". Raleigh (NC): SIGS Books-Alcatel Network Systems, Inc. juin 29, 1992. 71p.

[CUGY 83] CUGY, A. "*Organisation de l'Entreprise Moyenne. Initiation et Pratique*". Paris: Les éditions d'Organisation, 1983. 306p.

[DAVID 93] DAVID P.Y. "*Bluffez Votre Entourage avec Vos Connaissance Temps-Réel*". Dossier Temps-Réel, Bulletin de la Liaison de TRIBUNIX, 1993, Vol. 9, No. 47, p.2-5.

[DE VETTOR 91] DE VETTOR, P. "*Une architecture Logicielle à Objets pour la Conception d'Applications Industrielles Complexes*" Thèse Doct.: Université de France-Comté, 1991. 248p.

[DORSEUIL et al. 91] DORSEUIL, A. et PILLOT, P. "*Le temps Réel en Milieu Industriel: Concepts, Environnements, Multitâches*". Paris: Dunod, 1991. 296p.

[DOUMEING 91] DOUMEINGTS, G. "Méthodes pour Concevoir et Spécifier les Systèmes de Production". *Colloque International on CIM: Integration Aspects*, Bordeaux, France, juin 12-14, 1990. p.89-103.

[DOUMEING et al. 91] DOUMEINGTS, G. et VALLESPIR, B. "Techniques de Modélisation pour la Productique". *Proceedings of the 23rd CIRP International Seminar on Manufacturing systems*, Tome I, Nancy, France, juin 6-7, 1991. 10p.

[DUFFAU et al. 84] DUFFAU, B. et BLOCHE, E, "La Simulation des Unités de Production". *Intelligence Artificielle et Productique*, 1984, Vol. 1, No. 1, p.19-24.

[ERSCHLER et al. 85] ERSCHLER, J., FONTAN, G. et MERCE, C. "Consistency of the Dissaggregation Process in Hierarchical Planning". *Operations. Researches*, 1985, Vol. 34, No. 4, p.464-469.

[EVAN 88] EVAN, J.B. "*Structures of Discrete Event Simulation: An Introduction to the Engagement Strategy*". London: Ellis Horwood Limited, 1988. 279p.

[FERBER et al. 88] FERBER, J. et GHALLAB, M. "Problématiques des Univers Multi-agents Intelligents". *Acte des 2ème Journées du PRC GRECO Intelligence Artificielle*, Toulouse, France, 1988. Toulouse: TEKNEA, p.295-320.

[FRAMLING et al. 93a] FRAMLING, K. et HAMMARSTROM, L. "Object-Oriented Tool for Modeling, Simulation and Production Planning in Petrochemical Industries". *Actes de la Représentations par Objets*, Grande Motte, juin 17-18, 1993. p.169-180.

[FRAMLING et al. 93b] FRAMLING K. et HAMMARSTROM L. "A Distributed Heuristic Expert System for Simulation and Production Planning in Petrochimical Industries". *Proceeding of the Workshop "Knowledge-Based Production Planning, Scheduling and Control" of IJCAI'93*, Chambery, France, août 28- sept. 3, 1993. p.149-158.

[FRITSCHY 90] FRISCHY, D. "La Simulation Orientée Objet: Nouvelles Exigences en Matière de Simulation Informatique et Temporelle des Flux d'un Système Flexible et Discontinu de Production". *Colloque International sur Productique & Intégrations*, Bordeaux, juin 12-14 1990. p.587-598.

[GACHES et al. 93] GACHES, R., QUERENET, B. et VERNADAT, F. "CIMOSA: une Architecture pour l'Intégration dans les Entreprises Manufactutières". *Acte de Université d'Eté du Pole Productique Rhone-Alpes*, Aussois, France, sept. 13-17, 1993. 15p.

[GALISSON 94] GALISSON, Y. "NORMES: Sans Standarts, Point de Salut". *01 Informatique, Dossiers Spécials sur les Technologies Objets*, 1994, No. 1298, p. 40.

[GEHANI et al. 88] GEHANI, N.H. et ROOME, W.D. "Concurrent C++: Concurrent Programming With Class(es)". *Software-Pratice and Experience*, 1988, Vol. 18, No. 12, p.1157-1177.

[GERSHWIN 89] GERSHWIN, S.B. "Hierarchical Flow Control: A Framework for Scheduling and Planning Discrete Events in Manufacturing Systems". *Proceedings of the IEEE*, 1989, vol 77, No. 1, p.195-209.

[GIARD 88] GIARD, V. "*Gestion de Production*". Paris: Economica, 1988. 1068p.

[GOLDBERG et al. 83] GOLDBERG, A. et ROBSON, D. "*Smalltalk-80: the Language and its Implementation*". Massachusetts: Addison-Wesley, 1983. 715p.

[GOLDRATT et al. 86] GOLDRATT, E.M. et COX, J. "*Le But: L'Excellence en Production*". Paris: Afnor Gestion, 1986. 238p.

[GOMMA 84] GOMMA, H. "A Software Design Method for Real-Time Systems". *Commun. of ACM*, 1984, Vol. 27 No. 9, p.938-949.

[GONNET 76] GONNET, G.H. "Heaps Applied to Event Driven Mecanisms". *Commun. of ACM*, 1976, Vol. 19 No. 7, p.417-418.

[GOVINDAR 90 et al.] GOVINDAR, T., MCGINNIS, L.F., PLATZMAN, L.K. et MCGINISS, L.F. "Manufacturing Simulation Using Objects". *Proceeding of the 1990 Summer Computer Simulation Conference*, juillet 16-18, Calgary, Canada, 1990. p.219-224.

[GUILLARD 92] GUILLARD, S. *"Modélisation et Stratégies Auto-Organisatrices pour les Ateliers de Production Modernes"*. Thèse Ing.: Institut National des Sciences Appliquées de Lyon, 1992. 235p.

[GUO et al. 90] GUO, D., NORRIE, D.H. et FAUVEL, O.R. "Object-Oriented Flexible Manufacturing System Simulation". *Proceeding of the 1990 Summer Computer Simulation Conference*, juillet 16-18, Calgary, Canada, 1990. p.225-230.

[HAMICHI et al. 88] HAMICHI, S. et KEIFFER, J.P. "L'implémentation d'une Gestion de Production Informatisée: du Projet à la Mise en Route, les Progiciels et leurs installations, Conseils d'Organisation, deux Exemples Détaillés et Commentés". Paris: Editions du Moniteur, 1988, 248p.

[HENDERSON et al. 91] HENDERSON-SHELLERS, B. et EDWARDS, J.M. "Object-Oriented Systems Life Cycle". *Commun. of ACM*, 1991, Vol. 33, No. 9, p.142-169.

[HATLEY et al. 91] HATLEY, D.J. et PIRBHAI, I.A. *"Stratégies de Spécification des Systèmes Temps Réel"*. Paris: Masson, 1991. 368p.

[HAX et al. 85] HAX, A.A. et MEAL, H.C. "Hierarchical Integration of Production Planning and Scheduling". *Studies in Management Sciences*, 1975, Vol. I, 1975. p.53-69.

[HILL 93] HILL, D.R.C. *"Analyse Orientée Objets & Modélisation par Simulation"*. Paris: Addison-Wesley, 1993. 362p.

[HOLLINGER 87] HOLLINGER, D. "Les Langages Orientés-Objets en Spécification et Simulation des Systèmes Productiques". *Journées SIRRODIS 'Pratique de la Simulation en Production Discontinue*, Paris, juin 2-3, 1987, p.79-91.

[HOOGEBOOM et al. 93] HOOGEBOOM, B. et HALANG, W.A. "The Concept of Time in Software Engineering for Real Time Systems". *3rd International Conference on Software Engineering for Real Time Systems, IEEE Conference Publication No. 344*, 1993. p.156-163.

[IGL 89] IGL Technology *"SADT: un Langage pour Communiquer"*. Paris: Eyrolles, 1989. 326p.

[ISHIKAWA et al. 92] ISHIKAWA, Y., TOKUDA, H. et MERCER, C.W. "An Object-Oriented Real-Time Programming Language". *IEEE Computer*, Vol. 25, No. 10, oct. 1992. p. 66-73.

[JAIN et al. 90] JAIN, J., BARBER, K. et OSTERFELD, D. " Expert Simulation for On-Line Scheduling". *Commun. of ACM*, 1990, Vol. 33, No. 10, p.54-60.

[JAULENT 92] JAULENT, P. "*Génie Logiciel: les Méthodes*". Paris: Armand Colin éditeur, 1992. 294p.

[JAVEL 93] JAVEL G. "*L'Organisation et la Gestion de Production*". Paris: Masson, 1993. 328p.

[JEFFERSON 83] JEFFERSON, D. "Virtual Time". *Proceeding International Conference on Parallel Processing*, Silver Spring, Md., 1983. p.384-394.

[JULLIEN 91] JULLIEN, B. "*Simulation des Systèmes de Production*". Ecole Nationale Supérieure des Mines de St-Etienne, 1991. 172p. Support de Cours.

[JULLIEN 92] JULLIEN, B. "Un modèle Conceptuel pour la Simulation Orientée Processus". *2ème Journée Conférence FRANCOSIM*, Villeurbanne, juin 18 1992. 15p.

[KELLERT 92] KELLERT, P. "Définition et Mise en Oeuvre d'une Méthodologie Orientée Objets pour la Modélisation des Systèmes de Production". *Congrès Inforsid'92*, Clémont-Ferrand, mai 19-22, 1992. p.415-436.

[KRUEGER 92] KRUEGER, C. W. "Software Reuse". *ACM Computing Surveys*, 1992, Vol 25, No. 2, p.131-183.

[LABRECHE 90] LABRECHE, P. "Interactors: a Real-Time Executive with Multiparty Interactions in C++". *SIGPLAN Notices*, 1990, Vol. 25, No. 4, p.20-32.

[LEROUDIER 80] LEROUDIER, J. "*La Simulation à Evénements Discrets*". Paris: Hommes et Techniques, 1980.101p.

[LEVINE et al. 89] LEVINE, P. et POMEROL, J.C. "*Systèmes Interactifs d'Aide à la Décision et Systèmes Experts*". Paris: Hermes, 1989. 335p.

[LISSANDRE 90] LISSANDRE, M. "*Maîtriser SADT*". Paris: Armand Colin Editions, 1990. 219p.

[LOMOW et al. 90] LOMOW, G. et BAEZNER, O. "A Tutorial Introduction to Object-Oriented Simulation and SIM++". *Proceeding of the 1990 Summer Computer Simulation Conference*, juillet 16-18, Calgary, Canada, 1990. p.146-148.

[MAIRE 91] MAIRE, J.L. "*OLYMPIOS: un Modèle de Conception du Système d'Information d'une Entreprise Manufacturière-Application à l'Audit*". Thèse Doct.: Université de Savoie, 1991. 149p.

[MASINI et al. 89] MASINI, G., NAPOLI, A. COLNET, D. LEONARD, D. et TOMBRE, K. "*Les langages à Objets: Langages de classes, langages de frames, langages d'acteurs*". Paris: InterEditions, 1989. 583p.

[MATHON 92] MATHON, A. "*Appel d'Offre du Projet NADEGE: Renseignements Scientifiques*". Dépt. Stratégie du Développment de l'Ecole des Mines de St-Etinne, France, juin 1992. 22p.

[MCGREGOR et al. 92] MCGREGOR, J.D. et SYKES, D.A. "*Object-Oriented Software Development: Engineering Software for Reuse*". New-York: Van Nostrand Reinhold, 1992. 352p.

[MCLAREN et al. 88] MCLAREN, B.M., NEUSS, P.M. et GROOTE, O.D. "The Integrated Modeling Package: An Object Oriented Module for Manufacturing Simulation". *Proc. of the European Simulation Multiconference*, Nice, juin 1-3, 1988. p.231-236.

[MCPHERSON et al. 86] MCPHERSON, R.F. et WHITE, K.P. "A Management Control Approach to the Manufacturing Planning and Scheduling Problem". *Proceedings of a Symposium Held at the Nation Bureau of Standards on Real-Time Optimisation in Automated Manufacturing Facilities*, Gaithersburg, MD, jan. 1986. p.145-165.

[MENGA et al. 89] MENGA, G., MORISIO, M. et LO RUSSO, G. "*A Framework for Object Oriented Design and Prototyping of Manufacturing Systems*". Politecnico di Torino, Italy, 1989. 26p. Rapport du Dépt. Automatique et Informatique.

[MELESE 79] MELESE, J. "*Approches Systémiques des Organisation: vers l'Entreprise à Complexité Humaine*". Paris: Hommes et Techniques, 1979. 158p.

[MEYER 88] MEYER, B. "*Object-Oriented Software Construction*". Englewood Cliffs, N.J.: Prentice-Hall, 1988. 534p.

[MEYER 93] MERYER, B. "Systematic Concurrent Object-Oriented Programming". *Commun. of ACM*, 1993, Vol. 36, No. 9, p.56-80.

[MILLE 93] MILLE, Y. "Juste-à-Temps et Productivité: une Démystification Indispensable", *Revue Française de Gestion Industrielle*, 1993, Vol. 12, No.1, p.53-70.

[MIRY et al. 91] MIRY, S., MATHON, A. et VINCENT, L. "Evaluation Economique d'un Système de Production par Simulation de Politiques de Gestion en Présence d'Aléas". *3ème Congrès International Génie Industriel*, Tours, mars 20-22, 1991. p.635-645.

[MIRY et al. 92] MIRY, S., MATHON, A. et GIRARD, M-A "SIM'I: un Outil Pédagogique en Gestion de Production". *Congrès Inforsid'92*, Clémont-Ferrand, mai 19-22, 1992. p.553-562.

[MIRY 93] MIRY, S. "*Contribution à la Modélisation Globale d'Entreprise: Décision et Performances dans les Systèmes de Production*". Thèse Doct. Ing.: Institut National des Sciences Appliquées de Lyon, 1993. 306p.

[MIZE et al. 89] MIZE, J.H., BEAUMARIAGE, T. et KARACAL, C. "Systems Modeling Using Object-Oriented Programming". *Proc. of the 1989 International Industrial Engineering Conference.*, Norcross, GA, USA, avril 1989. p.13-18.

[MOIGNE 90] LE MOIGNE, J.L. "*La Modélisation des Systèmes Complexes*". Paris: EditionsDunod, 1990.178p.

[MONARCHI et al. 92] MONARCHI, D. E. and PUHR, G. I. "A Research Typology for Object-Oriented Analysis and Design". *Commun. of ACM*, 1992, Vol 35, No. 9, p.35-47.

[MORIN et al. 93] MORIN, P., GERBEAUD, M. et LAJOIE, M. "Ingénierie Simultanée: un Nouveau Cercle à l'AFITEP". *La Cible*, 1993, Vol. 46, p.18-19.

[MULKENS 93] MULKENS, H. "Les Nouvelles Organisations Productives". *Revue Françaice de Gestion Industrielle*, 1993, Vol. 12, No.3, p.5-30.

[NELSON 91] NELSON, M.L. "Concurrency & Object-Oriented Programming". *ACM SIGPLAN Notices*, 1991, Vol. 26, No. 10, p.63-72.

[NGUYEN et al. 92] NGUYEN, G.T. et RIEU, D. "Multiple Object Representations". *Proc. of the ACM Computer Science Conference*, Kansas City, mars 1992. p197-204.

[NGUYEN 93] NGUYEN, G.T. "SHOOD: Plate-forme pour la Conception Assistée". *Ingénierie des Systèmes d'Information*, 1993, Vol. 1, No. 3, p.393-414.

[NICOLAS 91] NICOLAS, P. "*Meijin++, Reference Manual*". Network Integrated Services, INC, Santa Ana, CA. 1991.800p.

[NICOLAS 93] NICOLAS, P. "Object-Oriented Simulation Using C++". *Borland International Conference*, San Diego, CA, USA, mai 1993. 21p.

[O'GRADY 86] O'GRADY, P.J. "*A Hierarchy of Intelligent Scheduling and Control for Manufacturing Systems". Proceedings of a Symposium Held at the Nation Bureau of Standards on Real-Time Optimisation in Automated Manufacturing Facilities*, Gaithersburg, MD, jan. 1986. p.15-30.

[OLUMOLADE et al. 90] OLUMOLADE, M., NORRIE, D.H. et FAUVEL, O.R. "Object-Oriented Integration and Simulation of Manufacturing". *Proceeding of the 1990 Summer Computer Simulation Conference*, juillet 16-18, Calgary, Canada, 1990. p.249-252.

[ORLICKY 75] ORLICKY, J. "*Material Requirements Planning: the New Way of Life in Production and Inventory Management*". New-York: Mc Graw-Hill, 1975. 292p.

[OUZOUT et al. 94] OUZOUT, Y., YE, X.J., VINCENT L. et JULLIEN B. "Designing and Implementing a Process-Oriented Simulation Environment". *International Conference on Concurrent Engineering and Electronic Design Automation*, Bournemouth, U.K., avril 7-8,1994. p469-474.

[PANWALKER et al. 77] PANWALKER, S.S. et ISKANDER, W. "A survey of Scheduling Rules". *Operation Research*, 1977, Vol. 9, No. 1, p.45 - 61.

[PEGDEN et al. 90] PEGDEN, C.D., SHANNON, R.E. et SADOWSKI, R.P. "*Introduction to Simulation Using SIMAN*". New-York: McGraw-Hill, Inc, 1990. 615p.

[PIERREVAL 90] PIERREVAL, H. "*Les Méthodes d'Analyse et de Conception des Systèmes de Production*". Paris: Editions Hermès, 1990. 62p.

[POPKEN 92] POPKEN, D.A. "An Object-Oriented Simulation Environment for Airbase Logistics". *Simulation*, 1992, Vol. 59, No. 11, p.328-338.

[PORTMANN 87] PORTMANN, M.C. "*Méthodes de Décomposition Spatiale et Temporelle en Ordonnancement de la Production*". Thèse d'Etat: Univ. de Nancy I, 1987. 315p.

[POUNTAIN 94] POUNTAIN, D. "The Chorus Microkernel". *Byte Magasine*, jan. 1994. p.131-136.

[PRIETO 90] PRIETO-DIAZ, R. "Damain Analysis: an Introduction". *ACM SIGSOFT Software Enginneering Notes*, 1990, Vol 15, No. 2, p.47-54.

[PRITSKER 86] PRITSKER, A.A. "*Introduction to Simulation and SLAM II*". USA: Systems Publishing Corporation, 1986. 839p.

[QUANG 89] QUANG, P.T. "MERISE/YOURDON". *Revue Génie Logiciel*, No. 15, 1989. p.22-39.

[RECHENMANN 88] RECHENMANN, F. "*Shirka: un Système de Gestion de Bases de Connaissances: Manuel de Référence*". Ecole Nationale Supérieure de l'Information et des Mathématiques Appliqués de Grenoble, juin 1988. 240p.

[RIEU et al. 91] RIEU, D. et CULET, A. "Classification et Représentation d'Objets". *Congrès INFORSID'91*, Paris, France, juin 4-7, 1991. p.85-107.

[ROCHELD 93] ROCHELD, A. et BOUZEGHOUB, M. " From Merise to OOM". *Ingénierie des Systèmes d'Information*. 1993, Vol. 1, No. 2, p.151-176.

[RODDE 90] RODDE, G. "*Les Systèmes de Production: modélisation et performances*". Paris: Hermés, 1990. 333p.

[ROLLAND] ROLLAND, C. "Indroduction à la Conception des Systèmes d'Information et Panorama des Méthodes Disponibles". *Genie Logiciel*, 1986, No 4, p.6-11.

[ROSE 92] ROSE, O. "*SIM Plus Plus, User Manual*". University of Karlsruhe, Germany, juin 1992. 55p.

[ROY 70] ROY, B. "*Algèbre Moderne et Théorie des Graphes*". Tome 2, Paris: Dunod, 1970. 759p.

[RUMBAUGH et al. 91] RUMBAUGH, J., BLAHA, M., PREMERLANI, W., EDDY, F. et LORENSEN, W. "*Object-Oriented Modeling and Designing*". Englewood Cliffs, N.J.: Prentice Hall, 1991. 354p.

[SCHERER 90] SCHERER, M. "Combat d'Idées Autour de la GPAO". *revue Industries Techniques*, 1991, No. 3, p.46-47.

[SCHIPER 86] SCHIPER, A. "*Programmation Concurrente*". Laussane: Presses Polytechniques Romandes, 1986. 295p.

[SHLAER et al. 92] SHLAER, S. et MELLOR, S. "*Object LifeCycles: Modeling the World in States*". Englewood Cliffs, N.J.: Prentice-Hall, 1992. 251p.

[SHOPIRO 87] SHOPIRO, J.E. "Extending the C++ Task System for Real-Time Control". *Proceeding and Additional Papers C++ Workshop*, Santa Fe, NM, novembre 9-10 1987. p.77-94.

[SNYDER 93] SNYDER, A. "The essence of Object: Concepts and Terms". *IEEE Software*, jan. 1993. p.31-42.

[STROUSTRUP et al. 87] STROUSTRUP, B. et SHOPIRO, J.E. "A Set of C++ Classes for Co-routine Style Programming". *Proceeding and Additional Papers C++ Workshop*, Santa Fe, NM, novembre 9-10, 1987. p.417-439.

[TARDIEU 89] TARDIEU, H. "De Merise à Metratech". *Genie Logiciel*, 1989, No. 15, p.40-52.

[TERSINE 88] TERSINE, R.J. "*Principles of Inventory and Materials Management*". New-York: Elsevier Science Publishing Co., Inc, 1988. 553p.

[THORIN 90] THORIN, M. "*Parallélisme: Génie Logiciel Temps Réel*". Paris: Dunod, 1990. 138p.

[THOMAS 92] THOMAS, L. "*Etude Comparée des Mécanismes de Synchronisation et de Communication dans les Langages à Objets pour le Parallélisme*". Thèse Doct.: Université de Nancy I, 1992. 244p.

[TRACZ 92] TRACZ, W. "Domain Analysis Working Group Report - First International Workshop on Software Reusability". *ACM SIGSOFT Software Enginneering Notes*, 1992, Vol 17, No. 3, p.27-34.

[TSICHRITZIS 88] TSICHRITZIS, D. "*Active Object Environments*". Centre Universitaire d'Informatique; Université de Genève, Genève Switzerland, 1988. 236p.

[VARHOL 94] VARHOL. P.D. "Small Kernels Hit It Big", *Byte Magasine*, jan. 1994. p. 119-128.

[VINCENT et al. 92] VINCENT, L., MIRY, S. et DIRARD M-A. "SIM'1: une Pédagogie de l'Intégration Productique". *1er Colloque Productique et Formation*, Marseille, oct. 27-28, 1992. p.145-150.

[VUILLEMIN 78] VUILLEMIN, J. "A Data Structure for Manipulation Priority Queues". *Commun. of ACM*, 1978, Vol. 21, No. 4, p.309-315.

[XIE 89] XIE, X.L. "*Contrôle Hiérarchique d'un Système de Production Soumis à Pertubations*", Thèse Doct. Université de Nancy I, 1989. 187p.

[WALDNER 90] WALDNER, J.B. "*Les Nouvelles Perspectives de la Production*". Paris: Dunod, 1990.165p.

[WALAS et al. 93] WALAS, M. et HUGUES, A.M. "Synthèse et Classification de Langages Parallèles". *Génie Logiciel*, 1993, No. 32, p.6-22.

[WAYNER 94] WAYNER, P. "Objects on the March". *Byte Magasine*, jan. 1994. p.139-150.

[WILLIAMS 90] WILLIAMS, S.A. "*Programming Models for Parallel Systems*". Chichester: WILEY, 1990. 170p.

[WRITE et al. 88] WRITE, T. et GRZYBOWSKI, R. "Object Based Simulation: Foundations and Implementation in Modula-2". *Proc. of the European Simulation Multiconference*, Nice, juin 1-3, 1988. p.193-198.

[WYATT et al. 92] WYATT, B.B., KAVI, K. et HUFNAGEL, S "Parallelism in Object-Oriented Languages: a Survey". *IEEE Software*, 1992, No. 11, p.56-65.

[YE 90] YE, X.J. "Automatisation de la Production: gamme d'usinage et ordonnancement." Mémoire de DEA: Institut National des Sciences Appliquées de Lyon, 1990. 92p.

[YE et al. 92a] YE, X.J., JULLIEN, B. et MATHON, A. "Scheduling by Object-Oriented Simulation". *Proc. of 3rd International Conference on Data and Knowledge Systems for Manufacturing and Engineering*, Lyon, France, mars 17-20, 1992. p.115-127.

[YE et al. 92b] YE, X.J. et MATHON, A. "Manufacturing Management System Simulation: an Object-Oriented Approach". *Proceeding of 2nd Beijing International Conference on System Simulation and Scientific Computing*, Pékin, octobre 20-23, 1992. p.30-34.

[YE 93] YE, X.J. "*Une Plate-Forme pour la Simulation Orientée-Objets: Expériences et Exigences*". Exposé dans le Groupe "Objet" du Pôle Productique, Grenoble, mars 21, 1993. 33p.

[YE et al. 93] YE, X.J., OUZROUT, Y. et MATHON, A. "Conception d'une Plateforme de Simulation de Système de Production par les Objets". *Proceeding of Industrial Systems Engineering*, Marseille, France, decembre 15-17, 1993. p.189-196.

[YE et al. 94a] YE, X.J., JULLIEN, B. et VINCENT, L. "Concurrent Object-Oriented Simulation for Manufacturing Applications". *Proceeding of Western Multiconference*, Tempe, Arizona, U.S.A., jan. 24-27, 1994. p.3-8.

[YE et al. 94b] YE, X.J., MIRY, S., MATHON, A. et VINCENT, L. "Modélisation et Simulation des Systèmes de Production par les Objets". *3rd Maghrebian Conference on Software Engineering and Artificial Intelligence*, Rabat, Maroc, avril 11-14, 1994.

[YE et al. 94c] YE, X.J., et FAVREL, J. "Production Flow Simulation". *4th International Conference on Data and Knowledge Systems for Manufacturing and Engineering*, HongKong, mai 2-4, 1994.

[YE et al. 94d] YE, X.J. et MATHON, A. "Objets-Oriented Process Simulation". *Proceeding of European Simulation Multiconference*, Barcelone, Espagne, juin 1-3, 1994.

[ZEIGLER 76] ZEIGLER, B.P. "*Theory of Modeling and Simulation*". New-York: John Wiley & Sons, 1976. 534p.

[ZEIGLER 89] ZEIGLER, B.P. "Hierarchical Modular Discrete-event Modeling in an Object Oriented Environment", *SIMULATION*, 1987, Vol. 49, No. 5, p.219-230.

[ZHAO 89] ZHAO, X.I. "*Contribution à la Construction d'un Système Expert d'Aide à la Conduite d'Ateliers Manufacturiers*". Thèse Doct.: Université de Valencienne, 1989. 189p.

ANNEXE 1

OBJET, SIMULATION ET PROGRAMMATION CONCURRENTE

ABSTRACTION ET ENCAPSULATION

Ces deux concepts sont complémentaires. L'abstraction se concentre sur la vue externe d'un objet et l'encapsulation empêche les clients de voir l'intérieur d'un objet, là où le comportement de l'abstraction est implémenté. L'encapsulation procure les différentes abstraction.

AGENT

Une entité physique ou abstraite, qui est capable d'agir sur elle-même et sur son environnement et qui dispose d'une représentation partielle de leur environnement, qui dans un univers multi-agents, peut communiquer avec d'autres agents, et dont le comportement est la conséquence de ses observations, de sa connaissance et des interactions des autres agents. On dit aussi un objet autonome qui a son propre contrôle).

ATTRIBUT (CHAMP)

Terme employé généralement dans le cadre de représentation des connaissances. Les attributs, outre un composant statique d'un objet auquel est associée une valeur, disposent d'informations complémentaires concernant la manière d'utiliser cette valeur.

CHANGEMENT DE CONTEXTE

Opération qui consiste à remplacer le contexte d'un processus qui utilisait le processeur par celui d'un autre processus qui doit devenir actif suite à l'arrivée d'événement. Dans la vie courante, c'est ce que nous posons une question sur un autre sujet.

CLASSE (MODULE, PACKAGE)

Il s'agit de la notion centrale de l'approche objet. Une classe est l'abstraction d'un ensemble d'objets qui ont des propriétés similaires. Les objets en sont la représentation concrète à l'exécution du système. Ce sont les instances de la classe. Une classe définit un ensemble d'objets par leur comportement et leur utilité, mais pas par leur implémentation. Elle comporte une représentation des données qui formeront l'objet (champs ou attributs) et des traitements qu'il est possible de leur appliquer (méthodes ou routines).

CLASSE ABSTRAITE (VIRTUELLE)

Une classe abstraite est une classe qui n'a pas pour vocation d'être instanciée, mais de décrire abstraitement un concept dans le but d'organiser la hiérarchie de classes ou de définir des méthodes et des variables qui s'appliqueront aux classes de plus bas niveau. Les classes abstraites servent ainsi de mécanisme de factorisation des caractéristiques des classes..

CONCURRENCE

Bel anglicisme qui signifie parallélisme. Ce faux ainsi est à proscrire.

CONTEXTE

Somme de toutes les informations qui permettent à un processus de savoir où il en est dans l'exécution d'une tâche.

COROUTINE

Une coroutine est un objet, avec son propre état d'exécution, qui peut être suspendu ou repris par ces deux routines membres: reprendre et suspendre. La routine *reprendre* est utilisée uniquement par les méthodes membres de la coroutine; elle active toujours la coroutine dont elle est élément et désactive la coroutine appelante. La routine *suspendre* est utilisée uniquement dans le corps de la coroutine; elle désactive cette coroutine et active une autre coroutine. Une coroutine s'exécute séquentiellement son comportement défini par une routine membre sans paramètre appelée *fonctionner*.

EVENEMENT

Un stimulus qui s'exécute sur le système, la manifestation de la vie du système. Il peut être externe (venant de l'extérieur du système) ou interne. Il sert de signalisation.

EXCEPTION

Deux sens très différents identifient le mécanisme d'exception. Le premier sens est propre à un langage de programmation (ADA, C++, EIFFEL, ...), où l'exception fait référence à un événement se produisant à l'exécution anormale de la méthode d'objet. L'autre sens rencontré dans les problèmes de représentation des connaissances, désigne le moyen de décrire des objets marginaux sans rapport avec les autres.

HERITAGE

Une classe peut être définie comme une extension ou une restriction d'une autre classe. La nouvelle classe est alors appelée "héritière" de l'autre. Chaque classe peut hériter de plus d'une classe et même plus d'une fois de la même classe. L'héritage peut donc être simple ou multiple. C'est une première forme de réutilisabilité.

HIERARCHIE

Rangement ou ordonnancement des abstractions. Les deux hiérarchies les plus connues sont: hiérarchie de classes (relation généralisation/spécialisation) signifiant "genre de..." et hiérarchie d'objet (relation agrégation/désagrégation) qui, lui, veut dire "faire partie de".

INSTANCIATION

Processus consistant à remplir le formulaire d'une classe générique ou paramétrée afin de produire une classe à partir de laquelle on peut créer des instances.

INTERRUPTION

La partie la plus délicate dans la conception d'un système à événements. En effet, les interruptions, signaux physiques donc de très bas niveau, ont des répercussions à un très haut niveau. Pour éviter le désagrément d'être interrompu, il est possible de masquer les

interruptions. Rien à voir avec un quelconque déguisement puisque le masquage correspond au refus de se laisser interrompre.

LIAISON STATIQUE (DYNAMIQUE)

Mécanisme pour lequel un envoi de message est mis en rapport avec une méthode. Il existe deux types de liaison. Dans un liaison dynamique, la méthode correspondant à un message est recherchée lors de l'exécution de l'envoi de message. En revanche, dans un liaison statique la recherche est la méthode est effectuée lors de la compilation des envois de message. Les liaisons statistiques, plus efficaces, ne peuvent être réalisées que pour des langages disposant d'un typage fort.

MESSAGE

Signal envoyé par un objet à un autre, qui demande à l'objet récepteur d'exécuter l'un de ses méthodes. Un message est constitué de trois parties: le nom du récepteur, la méthode qu'il doit exécuter, et les paramètres éventuels dont la méthode peut avoir besoin pour remplir sa tâche.

MESSAGE SYNCHRONE/ASYNCHRONE

Termes utilisés dans la programmation concurrente à objets. Pendent l'envoi d'un message synchrone, l'émetteur attend ou se bloque jusqu'à ce que le récepteur ait reçu le message. Dans le cas d'envoi d'un message asynchrone, l'émetteur continue son exécution après avoir envoyé le message, mais sans attendre que le récepteur le reçoive. Les avantages de l'envoi des messages synchrones sont le transfert bidirectionnel des informations, la compréhension, la sémantique, la synchronisation, l'indication des erreurs, etc., et les avantages de l'envoi des messages asynchrones sont la flexibilité, la capacité d'expression, le parallélisme, etc.

METACLASSE

La classe d'une classe; une classe dont les instances sont elles-mêmes des classes.

METHODE

Procédure décrite dans une classe, qui est exécutée lors de la réception d'un message par l'une des instances de cette classe. Les méthodes sont les composantes dynamiques des objets.

MODULARITE

Propriété d'un programme qui a été découpé en modules. Scinder un programme en module permet d'en réduire sa complexité.

MONITEUR

Un moniteur est une unité syntaxique analogue à un module il permet de regrouper des variables ainsi que les procédures agissant sur ces variables. Les variables moniteurs ne sont accessibles qu'à travers les procédures moniteur. En d'autre terme, un moniteur est une classe pouvant être exécutée tour à tour par différents processus, mais en plus, le moniteur assure l'exclusion mutuelle sur les procédures déclarées dans le moniteur.

MONOMORPHISME

Un concept dans la théorie des types, selon lequel un nom peut seulement désigner des objets de même classe.

MULTIPROGRAMMATION (MULTITRAITEMENT, TRAITEMENT REPARTI)

On appelle multiprogrammation le cas où les processus se partagent une machine, y comprise le processeur (il y a donc simultanéité globale et entrelacement des activités, mais pas simultanéité réelle à un instant); multitraitement le cas où les processus se partagent les mémoires, chacun ayant son processeur; traitement réparti le cas où chaque processus a sa machine et est autonome, mais relies aux autres par un réseau de communication.

OBJET (INSTANCE)

C'est une entité qui présente quelques comportements bien particuliers. Un objet se caractériser par un état (qui englobe toutes ses propriétés), un comportement (façon dont il agit et réagit) et une identité (ce qui le distingue des autres). La structure et le comportement d'objets similaires sont définis dans une classe commune.

OBJET ACTIF (CONCURRENT)/PASSIF (SEQUENTIEL)

Un objet actif est un objet qui possède les trois propriétés suivantes: 1) un objet thread qui est un code d'exécution qui s'exécute indépendamment des objets; 2) un état d'exécution qui contient les informations d'états nécessaires à l'exécution concurrente de l'objet; et 3) un contrôle de l'exclusion mutuelle qui permet d'exécuter une méthode sur une ressource et exclut que d'autres méthodes agissent sur cette ressource.

OBJET AUTONOME (CONTROLE)

Un objet autonome est un objet concurrent qui a une activité autonome.

ORDONNANCEUR (SCHEDULER, NOYAU DE SYNCHRONISATION)

Module logiciel chargé de gérer les tâches et les processus. On peut considérer qu'un ordonnanceur est un concentré du système d'exploitation dans lequel on retrouve le minimum vital. Il existe en général deux approche pour ordonnancer les tâches et les processus. La première est basée sur la notion d'événement, c'est-à-dire, le scheduler fournit des fonctions élémentaires permettant d'ordonnancer les événements dans le temps. La deuxième approche est basée sur la notion de processus: le modèle simulé est décrit par un ensemble de processus progressant dans le temps. Chaque processus est un objet concurrent dans ce deuxième approche. Il est compatible donc avec l'approche objet.

ORDONNANCEMENT

Méthode qui permet à tout moment de décider quel processus utilise quel processeur.

PARADIGME

Manière de penser qui dirige, consciemment ou inconsciemment les pensées et les actions. Les paradigmes sont essentiels car ils constituent un référentiel collectif concernant la pensée et l'action. Toutefois, ils représentent un obstacle majeur à l'adoption de nouvelle approches, foncièrement meilleures.

PERSISTANCE

C'est la propriété d'un objet selon laquelle son existence transcende te temps (l'objet continue à exister après la fin de l'exécution du programme qui l'a créé) et/ou l'espace (c'est-à-dire que

l'adresse de l'objet dans l'espace peut changer par rapport à son adresse de création). Cela est très important pour des systèmes qui s'exécutent sur un ensemble de processeurs distribués.

POINT DE VUE

Un point de vue est une interprétation de tout ou partie des données d'un objet. Ce concept est fondamental dans la problématique de la représentation des connaissances où diverses connaissances n'ont pas la même signification dans des domaines de discours différents.

POLYMORPHISME

Un concept dans la théorie des types, selon lequel un nom (par exemple une déclaration de variable), le polymorphisme permet désigner des objets de nombreuses classes différentes reliés par une superclasse commune. Ainsi, un objet désigné par ce nom peut répondre de différentes façons à un ensemble commun d'opérations.

PREEMPTION

Action qui consiste à interrompre un processus utilisant un processeur pour faire passer un autre processus. En général, la préemption est causée par le déclenchement d'un processus plus prioritaire que celui qui a la main.

PROCESSUS

Unité atomique d'exécution de tâches (Le processus est virtuellement un processeur). Afin de savoir où il en est dans l'exécution de sa tâche, le processus a un contexte. Un processus a aussi un état pour l'ordonnanceur: il peut être élu (on dit alors qu'il a la main sur le(s) processeur(s) sur le(s)quel il tourne), éligible (il attend la libération du processeur), bloqué (il attend un événement pour devenir éligible), terminé (le processus a accompli sa tâche).

PROCESSUS LEGER (POIDS PLUME)

La légèreté d'un processus d'un processus est proportionnelle à la légèreté de son contexte. Un processus léger est un modèle de communication interprocessus par une mémoire commune.

PROGRAMMATION CONCURRENTE (PARALLELE, SIMULTANEE)

La programmation concurrente est le nom donné aux notations et aux techniques de programmation exprimant que le parallélisme est possible; c'est-à-dire, l'utilisation d'outils et de techniques de mise en oeuvre de processus concurrents fait partie de la théorie des systèmes d'exploitation. Elle résout les problèmes de communication et de synchronisation.

PSEUDO-PARALLELE/QUASI-PARALLELE

Dans un schéma d'exécution pseudo-parallèle, le processeur est attribué aux différents processus par un noyau, sans l'intervention du programmeur. Par contre, dans un schéma d'exécution quasi-parallèle, le langage de programmation ne comporte pas de noyau et ne s'occupe pas de partager le processeur entre les différents processus. Dans le deuxième cas on utilise souvent le terme coroutine à la place du terme processus.

RENDEZ-VOUS

Le rendez-vous est un outil de synchronisation de plus haut niveau que le moniteur; en conséquence, il est nécessaire d'ajouter au mécanisme de rendez-vous proprement dit plusieurs

syntaxiques permettant de résoudre agréablement les différents problèmes de synchronisation qui peuvent se présenter.

SEMAPHORE

Un sémaphore est une variable entière ne pouvant prendre que des valeurs positives (ou nulles). Il fournit un mécanisme qui permet à des processus d'être faiblement synchronisés. En effet, une synchronisation fort enchaîne généralement une mauvaise utilisation des ressources. Un asynchronisme complet est impossible et induit une cacophonie.

SIMULATION

La simulation est une technique de modélisation qui consiste à reproduire le comportement dynamique d'un système sur l'ordinateur afin de mieux le connaître, de mieux maîtriser son évolution dans le temps dans un environnement donné.

SIMULTANEITE

Permet à des objets différents d'agir en même temps. Distingue un objet actif d'un objet qui ne l'est pas.

SURCHARGER

Associer plusieurs significations au même nom de méthode, ce qui permet à un seul message de déclencher différentes actions en fonction de l'objet qui le reçoit et des paramètres qui lui sont associés. Le langage sélectionne automatiquement la signification appropriée en regardant le type de l'objet récepteur et en vérifiant le nombre et le type de paramètres.

SYNCHRONISATION

Mécanisme qui permet de remettre les pendules à l'heure entre plusieurs processus. Le rendez-vous entre processus est un moyen de synchronisation.

SYSTEME SYNCHRONE/ASYNCHRONE

Un système est qualifié de synchrone si les tâches à effectuer sont fortement couplées. Dans un système synchrone, il existe en général un dispositif générant des événements cycliques à une fréquence fixée. Les autres événements (en nombre généralement très restreint) sont resynchronisés (ils sont traités cycliquement). Dans un système asynchrone, les tâches ont des conditions d'activation faiblement couplées. Cela conduit à des systèmes plus modulaires, moins monolithiques que les systèmes synchrones.

TACHE

Unité atomique de travail pour une machine. En général, dans le temps réel, les programmes sont composés de plusieurs tâches qui s'exécutent en parallèle. Une tâche cyclique est une tâche où le travail à faire est sempiternellement le même.

THREAD

Littéralement, il s'agit d'un fil. Ce fil est le fil conducteur de ce que l'on doit faire. Il y a aussi une notion de "séquentialité" dans ce terme qui rappelle le file de la vie d'une tâche. Ils permettent d'avoir plusieurs flots de contrôles dans le même espace d'adressage. On parle de processus légers; ils ont leurs propres activités et ressources (piles, données).

TRAITEMENT EN PARALLELE

Technique de programmation où plusieurs activités se déroulent en même temps. Lorsqu'on l'implémente en se basant sur du logiciel, l'ordinateur simule des activités parallèles en passant rapidement de l'une à l'autre. Lorsqu'on l'implémente en se basant sur du matériel, plusieurs unités de traitement prennent en charge différentes parties du programme et les traitent réellement en parallèle.

TYPAGE

C'est le fait d'imposer la classe d'un objet de façon à empêcher les objets de différents types d'être échangés.

ANNEXE II

PROGRAMMES EN TETES DES CLASSES D'OBJETS

Fichier "THREAD.HPP"

```
****************************** THREAD.HPP ***********************************
```

Description

This structure is the abstract base class for all schedulable objects (or processes). It contains the data structure and member function that manipulate registers value and the stack parameters.

Functionalities:

1- Implement scheduling decisions and implementation for all concrete process classes derived from Thread.
2- Define data structures to manipulate stack parameters: stack pointers and base pointers.
3- Advance the virtual clock: clock
4- Perform context switch between Thread objects (processes).

```
class Thread : public Item {
      friend class Process;
      friend class Tracer;
      friend class Simulation;

public:
     Thread(const char*);         // creates a thread and initializes its signature with a signature (string).
 virtual         ~Thread();       // destructor.
     int        priority() const; // returns the priority of 'this' thread.
    void        priority(int);    // sets the priority level of this thread.
    void        exits(long);      // overwrites the standard library function ::exits (int) for coroutines.
 virtual void   store(FILE&);     // saves "this" thread into a file object.
 const char*  get_key() const;    // computes the identifier (signature) of 'this' thread.
     void     spawn();            // initializes the envionment of "this" thread by performing a context switch
 static TIME  get_clock();        // returns the virtual time of the scheduler.

protected:
 static Thread*  current;     // the thread currently active.
 static void*    basePtr;     // common stack base pointer for all the coroutines.
 static PQueue* scheduler;    //polymorphic scheduling structure (priority queue).
    DynArray* slaves;         // pointer to a dynamic array of threads blocked by 'this' thread.
 enum States {
      KILLED =1,         // the thread is inative (non-opérationel, crée).
      CONTROL =2,        // the thread is currently loaded on the stack (élu, en cours).
      IDLE =4,           // the thread stay inative until the scheduler or another thread wakes it up (bloqué).
      SCHEDULED =8 // the thread is scheduled to take control at a predefined time.
      }state;                // state of the thread.
 int       references;   // number of references to 'this' thread currently inserted in the sheduler.
 const char*   key;      // 'this' thread identifier (signature).
 TIME releaseTime;  // the time 'this' thread is reactivated (scheduled) to take control of the stack.
```

```
 bool      switch_out(); // saves the portion of the stack that supports the thread into buffer.
 void      schedule();   // pops the nexr thread from the scheduler and advances the virtual clock
                         // if necessary.
 virtual double rank() const; // computes the rank value of 'this' thread.
 virtual long   run();        // executes 'this' thread body.
 static void    process_err(const char*); // creates a processErr exception with the attribute.

private:
      Thread(const Thread&);              // copy constructor
     Thread& operator=(const Thread&); // member-wise assignment constructor

 static Thread* root;                 // pointer to the root Thread ( ::main(int argc, char **argv)) )
 static TIME   clock;                 // simulation time.
   void*  buffer;    // memory block to save the portion of the stack associated to 'this' thread.
   size_t size;              // size in bytes of the above buffer (Thread::buffer).
   jmp_buf env;              // buffer containing the thread environment (value of the registers).
   int    prty;              // 'this' thread's priority level.
 static void   (*action)(Thread*); // pointer to a function defining the tracer routine.
   void   switch_back(); // restores the stack frame associated with "this" thread from memory.
};    // THREAD_HPP
```

Fichier "PROCESS.HPP"

```
//************************ PROCESS.HPP ************************************
```

Process of a discrete event simulation model

Description

Processes are specialized, non pre-emptive threads that have a time generator and a set of variables (info). The array 'info' contains data or parameters that represent critical information about the process. This array can be used, for instance, to visualize the process. Processes communicate through a basic master/slave protocol where a process A (known as slave process) depends on another one to change its state. More sophiticated protocol, queues, are used by QNode objects.

```
class Process : public Thread {
public:
    enum     Mode {QUEUE,      // first-in-first-out.
                   STACK,      // last-in-first-out.
                   PRIORITY }; // the transaction are inserted according to their rank values.
                               // mode of item insertion in the queue.
    operator Generator*() const;  // returns a pointer to the timer generator (hatcher).
    void     block(Thread* =0);  // blocks "this" process until a thread reactivates it.
    void     set_generator(const Generator&); //initializes the time hatcher with a generator object.

protected:
    double* info;                 // array of parameters produced by the process (statistics).
    void    allocate_info(short); // initializes the attribute info.
    void    pending_one(Thread** T, ushort n =1); // blocks "this" process until at least one of the
                                                  // thread of the array T[n] is killed.
    void    pending_all(Thread** T, ushort n =1); // blocks "this" process until all the threads of the
                                                  // array T[n] are killed.
    void    delay (TIME t =0); // blocks "this" process for t units of time and reschedules it for time:
                               // (clock += t).
    long    preempts();        // blocks the process currently in control until "this" process is killed.
```

```
    virtual long run();                // executes the coroutine associated to this thread
    Process(const char*, Generator* =0);   //creates a process with a signature and a time generator
  virtual   ~Process();                //destructor
private:
    Generator* hatcher;   //time-advance generator
    Process(const Process&);  //copy constructor
    Process& operator=(const Process&);    //member-wise assignment constructor
};    //PROCESS_HPP
```

Fichier "MACHINE.HPP"

********** MACHINE.HPP **********

Single input/output queue machine representation for job-shop workshop simulation models.

Description

A machine object acts as an automous object that receives (consumes) items from its input queue, transforms the transactions and puts (produces) them into its output queue. A machine is characterized by six kinds of attributes: structural, contingent, instantaneous, event, historic and statistic. Structural attributes characterize fundamental aspects of machines such as the structure and the relationship with other objects. Contingent attributes describe the production context. Instantaneous attributes mean current state and their evolution depends on event attributes. Historic attributes permit to keep more or less the detail of the past. Statistic attributes are computation results that permit to compact historic attributes to obtain machines functioning laws. An automous object contains four types of methods: accessors, events generators, transformations (computations and processes) and tests. The use of the methods is defined in the behavior method run of active objects.

```
class Machine :: public Process {
friend class Transporteur;
friend class Station;

protected:
  double x, y;                          // machine position in the workshop.
  double machineValeur, initValeur;  // current and initial machine value.
  double prixUnitaire;                  // unit cost of utilization.
  int   capaciteEntree;                 // size of input queue.
  int   capaciteSortie;                 // size of output queue.

  Mode modeAmont;                       // input queue character (management mode).
  Mode modeAval;                        // output queue character.

  enum State {    non-engage,       // machine is in non-active state.
                  disponible,       // machine is in free state.
                  pret-a-charger,   // machine is in ready to load item (prepared) state.
                  productive,       // machine is in productive state.
                  pret-a-decharger, // machine is in ready to unload item state.
                  preparation,      // machine is in preparation state.
                  panne }           // machine is in breakdown state.
          etat;                     // machine state.
  TIME  tempsReparation;            // reparation time of machine.
  TIME  tempsPreparation;           // preparation time of machine.
  TIME  tempsUtilisation;           // operative time of machine.
  TIME  tempsEnPanne;               // Breakdown time of machine.
```

```
  TIME  duree;                    // production (simulation) time

#ifdef M_SUIVI
  int   stock;                    // number of parts in the input queue
  int   counteur;                 // number of manufactured parts by this machine.
  int   ordreLance;               // number of dispatched batches in the workshop.
  int   ordreTermine;             // number of terminated batches.
  int   lotDebutDate;             // Dispatched date of the batch.
  int   lotFinDate;               // Termined date of the batch.
#endif

#ifdef m_STATISTIQUE
  double tauxRebuts;              // machine scrap rate.
  double tauxOccupation           // machine utilization rate.
#endif

  Panne* panne;                   // pointer to breakdown treatment object.
  Palette* palette;               // pointer to preparation process.
  Transporteur ** transport;      // pointer to compatible associated transporters.
  Generator* loi;                 // time hatch generator

  SHOW(
  void show() const;)             // shows the value of machine attributes en mode text.

public:
  SortQ* entrée;                  // input queue.
  SortQ* sotie;                   // output queue.
  ResQ* ressource;                // resource queue.

  Part** reference                // reference candidate of parts that this machine can transform.
  Part* piece;                    // current part.
  Batch*  lot;                    // current batch.

  Machine(SortQ*, SortQ*, const char*, Generator* = 0);     // constructor
  ~Machine();                                               // destructor.

  double rank() const;                    // method for computation machine priority.
  void set_position(double, double ) ;    // set machine position.
  void set_stock(int s)             { stock = s;}
  void add_reparation(TIME r)       { tempsReparation+=r;}
  void add_panne(TIME p)            { tempsEnPanne += p;}
  void add_utilisation(TIME u)      { tempsUtilisation += u;}
  void add_compteur (int t)         {counteur += t;}
  void set_prix_unitaire(double uu) { prixUnitaire = uu;}
  void set_taux_rebut (double r)    { tauxRebuts = r;}
  void set_loi(Generator* g)        { loi = g;}
  void set_palette(Palette* p)      { palette =p;}
  void set_transport(Transporteur* t) { transport = t;}
  void machine_valeur_init(double v) { machineValeurInit = v;}
  void remise_counteur ()           { counteur = 0;}

  int   nombre_piece_fabrique() const { return counter;}    // If we remove the keyword const in the
  int   nombre_piece_stock() const    { return stock;}      // folowing methods, we can define other
  TIME  temps_reparation() const  { return tempsReparation;} // methods that read or write attribute
  TIME  temps_preparation() const { return tempsPreparation;} // value of implicated objects.
  TIME  temps_utilisation() const { return tempsUtilisation;}
  double taux_utilisation() const { return tauxRebut; }
```

```
  double taux_rebuts() const          { return tauxRebuts;}
  double prix_unitair() const         { return prixUnitaire;}
  double get_x() const                { return x;}
  double get_y() const                { return y;}
  Point& get-position();              // get machine position.
  TIME   temps_panne() const          { return tempsEnPanne;}
  Generator*  loi_panne() const       { return law;}
  Palette*    get_palette() const     { return palette;}
  Transporteur* transport() const { return transport;}
  double valeur_rester() {
          machineValeur = initValeur - prixUnitaire*tempsUtilisation;
          return machineValeur;
  }
  ...

  void   store(File&);                // saves the value of machine attributes into a file.
virtual void   next_fromQ();          // choice of next batch to transform.
virtual void   next_toQ();            // choice of next machine to transform this batch.
virtual Batch* choisir_lot();         // choice of next part.
virtual choisir_mode (Mode);          // choice of selection mode of parts in the input queue.

  double nombre_moyen_attente(); // computes and returns mean number of waiting parts.
  double temps_moyen_attente ();  // computes and returns mean waiting time.
  double taux_rebuts ();          // computes and returns
  double taux_occupation();       // computes and returns occupied rates.

  #ifdef m_SEMI_PERSISTENT
  void getf( File&);              // restores the content of the machine from a file.
  void putf(File&);               // stores the content of machine into a file.
  #endif

protected:
  Part* recevoir();               // receives a part from input queue.
  void transformer();             // transforms a part.
  void preparer();                // throws a preparation process for a batch.
  void  expedier (Part*);         // sends a part into output queue.
  void traiter_panne();           // throws a beakdown traitment process.
virtual long run();          // this routine must be redefined in derived class. Machine::run() performs
                             // only recevoir(), transformer() and expedier () methods. There is no
                             // decision implication in the machine behavior. So users have to overload
                             // their own decisions activities predefined or redefinied in derived objects.
private:
  void init();                    // initialization method for the constructor.
#ifndef m_SIMULATION_SIZE
  Machine (const Machine&);       // copy constructor.
  Machine& operator= (const Machine&); // member-wise assignment.
#endif
};        // MACHINE_HPP
```

Fichier "PART.HPP"

********** PART_HPP **********

Transaction part representation (passive objects) for workshop simulation models.

Description

The transaction units circulated in a simulation model are constituted of products, batches (job orders) and parts. The bill of material defines the link between products and its parts. In the presimulation module, we can define some methods to deduct from the product the structural and contingent attributes of job orders and parts. We choice to implement these transaction items as passive objects. So part is a passive object that describes the structure (process plan) that will be manipuleted by active objects in the production system.

```
class Part : public Item {

friend class Machine;
friend class  Station;
friend class Transporteur;
friend class Palette;
friend class Panne;

protected:
  TIME   tempsTot,        //time for scheduling
         tempsTard;
  TIME   tempsEntre,      //entry time for executing
         tempsCommence// effective start time
  int    position;        // position in the plan

public:
  enum State {
         non-active,
         stock,
         attente,
         transport,
         production,
         arret} etat;     // part state.
  double priorite;        // part priority.
  GSlist* gamme1;         // principal process plan.
  GSlist* gamme2;         // second process plan.
  TIME   tempsCycle,      // parameters for the statistics
         tempsAttente;
  double valeurInitial,   // raw material cost.
         valeurCourant,   // current part value.
         valeurAjoute;    // added value by the tranformation.

  Part (const char*,  GSlist* = NULL, double = 0.0);
                  // constructor with the part name, part processing plan and priority

  void set_nom(char* n)              {name = n;}
  void set_position(int n)           {position = n;}
  void set_temps_tot (TIME t)        {tempsTot = t;}
  void set_tempsTard (TIME d)        {tempsTard = d;}
  void set_temps_attente (TIME a)  {gamme1(position)->tempsAttente = a; }
  void set_temps_cycle(TIME t)     {gamme1(position)->tempsCycle = t;}
  void set_preparation_time(TIME) (gamme1(position)->tempsPreparation = t;}
  void set_gamme1(GSlist* t) {gamme1 = t;}
  void set_gamme2(GSlist* t) gamme2 = t;}

  double rank() {return (double) (priorite +temps_opeatoire(position));}
```

```
        // ici, on peut installer de diffentes methodes de gestion avec cette methode (fonction)

  GSlist* gamme1() const {return gamme1;}
  GSlist* gamme2() const {return gamme2;}
  char* get_nom() const {return nom;}
  int  get_position() const {return position;}

  TIME temps_operatoire (int position) const
                {return (gamme1->operator[](position)).tempsOperatoire;}
  TIME temps_operatoire (int position) const
                {return (gamme2->operator[](position)).tempsOperatoire;}

  TIME temps_attente (int position) ;      // waiting time of part in a principle operation
  TIME temps-attente (int position) ;      // waiting time of part in a secondairy operation

#ifdef m_SUIVI
  TIME temps_suivi();              // returns used simulation time.
  void add_temps-operatoire();     // adds service time to conserve operation time.
  TIME temps_a_lancer();           // computes release time of part.
  TIME temps_attente();            // computes total waiting time.
  TIME temps-attente-secondaire ();// computes total waiting time if we use SGlist.
#endif

  void set_tempsEntre (TIME t) { tempsEntre = t;}
  TIME get_tempsEntre() const {return tempsEntre;}
  TIME get_tempsCommence () const {return tempsCommence;}

  void store(File& part);          // saves the value of part attributes into a file.

  #ifdef m_SEMI_PERSISTENT
  void getf( File&);               // restores the content of the machine from a file.
  void putf(File&);                // stores the content of machine into a file.
  #endif

  SHOW(
  void show() const;)              // shows the value of part attributes en mode text.

private:
#ifndef m_SIMULATION_SIZE
  Part (const Piece& p);           //shallow copy constructor
  Part& operator= (const Part& p); // member-wise assignment
#endif

};                  // PART_HPP
```

Exemple Simplifié de la Définition du Comportement d'une Machine

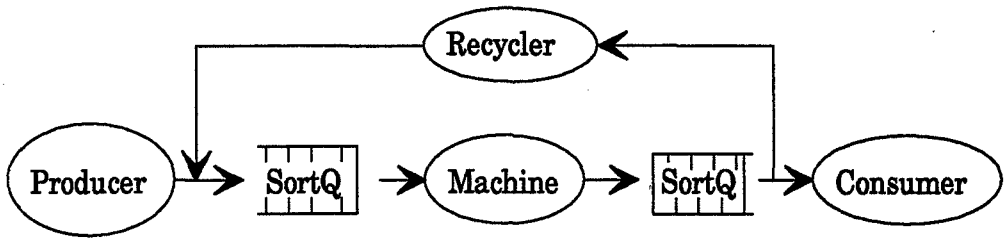
********** MACHINE_CPP **********

```
long Machine::run() {

  for(;;) {
  Part* piece = (Part*) recevoir();
  TIME t, tt;
  t = Thread::get_clock();
  piece->tempsEntre = t;
  TIME n = piece->tempsTot - t;
  if (n > 0.0)        //rupture d'approvisinnement
```

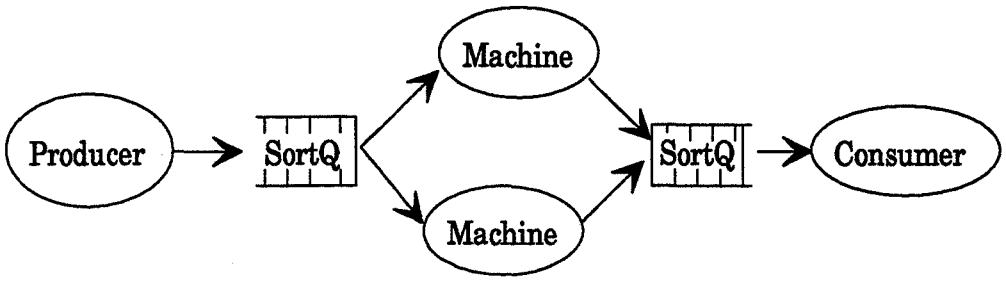
```
            block(this);
            // ici, on peut ajouter des service de Palette

    tt = piece->tempsAttente = Thread::get_clock() - piece->tempsEntre;
    piece->set_temps_attente (tt);

#ifdef m_PALETTE                        // on definit le processus preparation
    if (machine.conteur == 0) {
            palette->timer()
            tt = palette->(operator TIME() const);
            piece->set_preparation_time(tt);
            add_temps_preparation();
    }
#endif

    int p = piece->get_position();
    TIME tempsOperatoire = piece->temps_operatoire(p);
    delay(tempsOperatoire);
    add_counteur();
//  piece->priority += 1000.(piece-> -t);  on remet à jour la priorite de la piece.

    add_temps_operatoire (tempsOperatoire);

    (piece->position)++;                                // avancer la position de la gamme.
    piece->tempsCommence = Thread::get_clock();     // remettre le temps d'attente.

#ifdef m_TTRANSPORT
;                                       // on definit ici les interactions avec les transporteurs.
#endif

    expedier (piece);
    }

#ifdef m_PANNE
;                               // on definit ici le traitement de panne (non-pre-emptiple).
#endif
    return TRUE;
} //MACHINE_CPP
```

Modèle 1: modèle pour tester les méthodes *recevoire*, *transformer* et *expédier*. Les objets SortQ, Producer et Consumer sont des classes prédéfinies par Meijin++.

Modèle 2: modèle pour modéliser des opérations de retouche. L'objet Recycler est une instance de la machine.

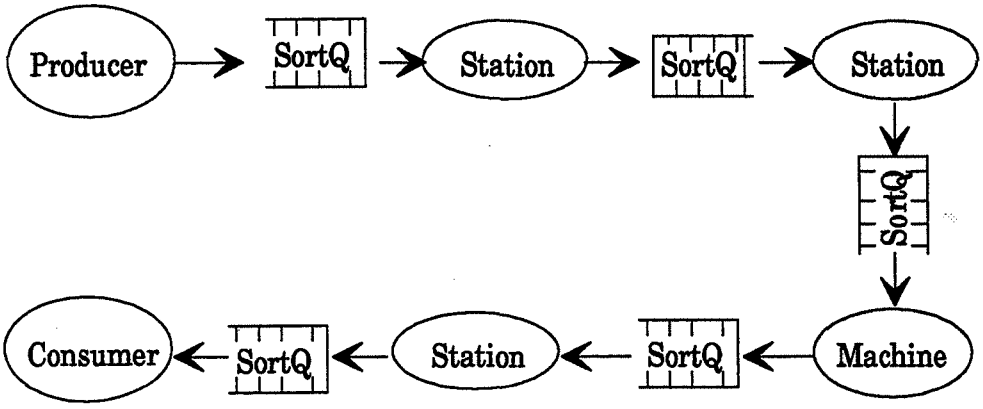


Modèle 3: deux machines identiques qui travaillent en parallèle. Le but est de définir des règles de gestion de pièces (décisions locales) dans la file d'attente.

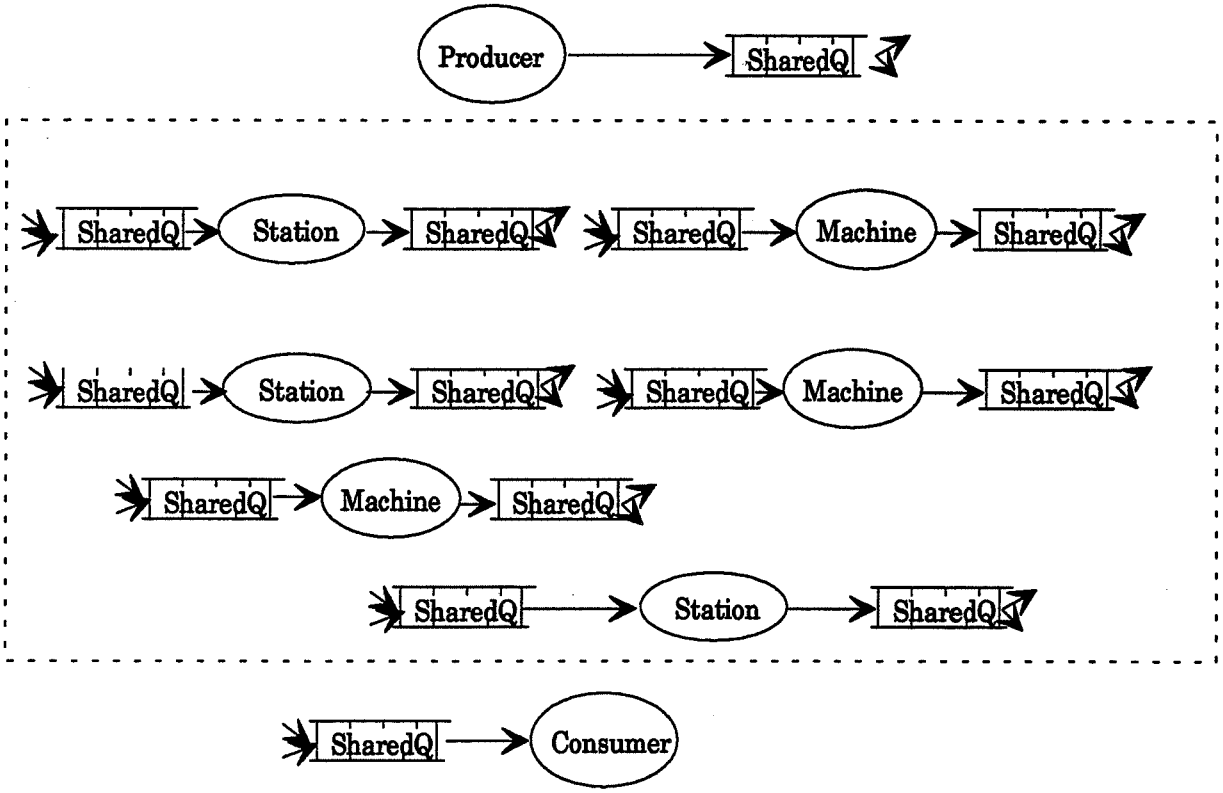


Modèle 4: une simplification du modèle 3. Les deux machines sont remplacées par un objet d'un haut niveau Station (encapsulation).

Modèle 5: atelier de flow shop, une extention de modèle 1.



Model 6: atelier de job shop sans des opération de transport.



Model 7: modèle pour modéliser la gestion des ressources partagées.